UNIVERSIDAD POLITÉCNICA DE MADRID

FACULTAD DE INFORMÁTICA

TRABAJO DE FIN DE MASTER

MASTER UNIVERSITARIO EN SOFTWARE Y SISTEMAS

Estudio preliminar acerca del uso de protocolos y actos comunicativos FIPA en el sistema COMPUTAPLEX

AUTOR: PAULO GUERRA

TUTOR: OSCAR DIESTE

MADRID 2014

Resumen en castellano

Este trabajo corresponde con la implementación de componentes software dentro de la Plataforma COMPUTAPLEX, la cual tiene como objetivo facilitar a los investigadores la realización de tareas del proceso experimental de ingeniería de software. Uno de los aportes a esta plataforma tecnológica corresponde con el desarrolló de los componentes necesarios para la recuperación de datos experimentales disponibles en diversas fuentes de datos, para ello se hizo uso de un mecanismo capaz de unificar la extracción de información de MySQL, ficheros excel y ficheros SPSS. Con ello diferentes grupos de investigación asociados pueden compartir y tener acceso a repositorios experimentales que se mantienen tanto de manera local como externa. Por otra parte, se ha realizado un estudio de la tecnología de agentes en la que se describe sus definiciones, lenguajes de comunicación, especificación FIPA, JADE como implementación FIPA y parser XML. Además para este trabajo se ha definido e implementado una ontología de comunicación entre agentes, la misma que fue diseñada en la herramienta Protégé. En lo que se refiere al desarrollo de componentes se hizo uso de una amplía variedad de tecnologías que incluye lenguaje de programación Java, framework JADE para el desarrollo de agentes, librería JENA para manejo de ontologías, librería SAXParser para lectura de archivos XML y patrón de diseño Factory. Finalmente se describe la metodología de trabajo utilizada en el proyecto, la cual por medio de la realización de varios ciclos iterativos permitió obtener prototipos que poco a poco fueron cubriendo las necesidades del producto software.

Palabras clave

agentes inteligentes, lenguaje KQML,FIPA,JADE,Parser XML, COMPUTAPLEX, Ontologías.

Abstract

This work relates to the implementation of software components within the platform Computaplex, which aims to enable researchers to conduct experimental software engineering process tasks. One of the contributions to this platform technology corresponds to the development of components which are necessary for the recovery of experimental data available in different data sources, to archive this goal a mechanism able to unify the extraction of information from MySQL, Excel and SPSS files was made.

Therefore, associated research groups can share and access experimental repositories that remain both locally and externally. Moreover, it has been conducted a study of agent technology in its definition is described, languages communication, FIPA, JADE and FIPA implementation and XML parser. In addition to this work, it has been defined and implemented an ontology for communication between agents, the same as was designed in the Protégé tool. In what refers to the development of components, a wide range of technologies have been made which includes Java programming language, framework JADE for agent development, JENA library for handling ontologies, SAXParser for reading XML files and Factory design pattern. Finally, describing the work methodology used in this project, which through the implementation of several iterative cycles allowed to obtain prototypes were gradually meeting the needs of the software product.

Keywords

intelligent agents, KQML language, FIPA, JADE, XML Parser, Computaplex, Ontology.

Índice general

Índice de figuras

Índice de cuadros

Agradecimientos

Quiero agradecer primero a Dios Jehová por ayudarme en todo momento de mi existencia, por darme su bendición en los momentos más difíciles, en los que su voluntad se ve reflejada en la culminación de este trabajo.

A mis familiares en Madrid: Mercedes, Juan, Gloria, Gina, Cindy y Octavio por brindarme su apoyo y acogerme como un integrante más de su familia, alentándome a seguir a delante y no desmayar.

A mis familiares y amigos en Ecuador, por darme el ánimo y estar pendientes de cómo me encuentro.

Mi más sincero agradecimiento a mi tutor Oscar Dieste quién con su paciencia, sabiduría y buena voluntad me ha colaborado en todo el proceso de aprendizaje para poder alcanzar el objetivo propuesto. Así también agradezco a mis profesores de la UPM quienes me guiaron y estuvieron prestos para ayudarme.

Finalmente mi agradecimiento a la Secretaria de Educación Superior, Ciencia Tecnología e Innovación SENESCYT, por la beca recibida para cursar mis estudios de postgrado.

Dedicatoria

Este trabajo está dedicado a mi familia quienes son los pilares fundamentales de mi existencia, Mi padre: Segundo Elías Guerra Vaca con su apoyo incondicional, me ha guiado para superar los obstáculos, y con su presencia me ha dado la seguridad para afrontar las dificultades. A la mejor madre del mundo: Mariana Isabel Terán Vaca quien con su amor y sabiduría me ha dado los mejores consejos en la vida, para fortalecer el espíritu, y llenarla de positivismo. A mis hermanos Juan Carlos y Marcelo Guerra quienes con su alegría han contagiado mi vida de felicidad, así también en toda circunstancia han estado prestos ayudarme.

Dedico también este trabajo a mi abuelita Ana Luisa Vaca, a todos mi tíos, tías, primos, primas y familiares con quienes he podido compartir momentos felices y circunstancias adversas, pero a pesar de todo sé que estamos juntos para seguir apoyándonos.

Capítulo 1

Introducción

1.1. Área en la que se encuadra el trabajo de Tesis de Fin de Máster

El presente trabajo se encuadra dentro del ámbito de la Ingeniería de Software Experimental (ISE), la cual es una disciplina de la Ingeniería de Software (IS), que promueve la utilización del método científico experimental como medio para la generación de conocimiento sobre el proceso de desarrollo de software. Surge como una actividad para conocer las causas por las cuales se producen determinados resultados dentro de dicho proceso. A través de la experimentación en IS es posible evaluar de manera objetiva distintas tecnologías para la construcción de sistemas software identificando y comprendiendo las relaciones entre las variables relevantes del fenómeno de estudio[25].

Para Pfleeger, experimentar con la construcción de software nos permitirá aumentar la comprensión de los factores o variables que hacen "al software bueno y cómo hacer software bien"[25]. El proceso experimental involucra un conjunto de actividades, productos, técnicas y herramientas necesarias para alcanzar los objetivos de investigación[5]. De esta manera un proceso básico de experimentación estará formado por las actividades: planteamiento, diseño experimental, operación, análisis estadístico y reporte de resultados[31].

Un experimento es una investigación formal, rigurosa y controlada donde se manipulan variables bajo estudio (método de inspección, técnica de pruebas, experiencia de desarrolladores entre otras). Los experimentos permiten averiguar los efectos que se producen en las variables respuesta (eficacia, eficiencia o productividad, etc.) cuando se realizan variaciones en los valores de las variables de estudio[31].

Actualmente, no existen evidencias sobre la adecuación, límites, cualidades, costes y riesgos de las tecnologías que se emplean en el desarrollo de software[13]. Aún los resultados de la aplicación de una determinada tecnología para la construcción de software es impredecible[31]. A menudo se adoptan nuevas tecnologías sin contar con evidencia convincente de que serán efectivas[16]. Por lo que la experimentación dentro de la Ingeniería de Software persigue que el desarrollo de software sea una actividad predecible científicamente, en la que se pueda comprender la relación existente entre los proceso de producción de software y los productos generados.

1.2. Necesidad de replicaciones Experimentales

La experimentación, como toda actividad realizada por personas está sujeta a error y desviaciones. Los investigadores pueden introducir sesgo en todas las actividades relacionadas con un experimento, desde el planteamiento de la hipótesis hasta el análisis de resultados[5].
Para asegurarnos que los resultados obtenidos en un experimento son fidedignos, es necesario replicar el experimento. Un experimento por si sólo puede proporcionar muy poca información sobre el fenómeno estudiado, por lo que es necesario repetirlo para ganar confianza en los resultados que arroja, atenuar la influencia de las subjetividad o para discernir las condiciones bajo las cuales dichos resultados son válidos y no fruto del azar.

La replicación en Ingeniería de Software consiste en la repetición de un experimento[5] con el objetivo de verificar que se observan los mismos efectos que ocurrieron en experimentos previos.

Para que un nuevo investigador o grupo replique un experimento con éxito se requiere usar distintos mecanismos e instrumentos que permitan la comunicación entre investigadores. La interacción entre los involucrados en la transferencia de conocimiento experimental puede darse por medios presenciales o no presenciales[30].

La interacción y documentación necesaria para desarrollar la replicación de experimentos muchas veces mantiene inconsistencia, incompletitud de detalles sobre los diseños, instrumentos, procedimientos con los cuales fue ejecutado cierta investigación,etc. Por ello se hace necesario contar con una infraestructura de soporte tecnológico de tal forma que grupos de investigadores puedan disponer de un medio de transferencia de conocimiento.

Una de las formas en las que se puede apoyar la replicación de experimentos es mediante infraestructura y herramientas específicas para la tarea. Las infraestructuras se suelen centrar en el material del experimento o cómo administrar dicho material. Ejemplos de este tipo son los repositorios de información experimental[5].

1.3. La plataforma COMPUTAPLEX

Las infraestructuras tecnológicas propuestas hasta el momento presentan ciertas deficiencias para apoyar al investigador durante el proceso de experimentación, no solo respecto a un experimento en particular, sino también en el tratamiento de familias de experimentos desarrollados en conjunto por grupos de investigación.

COMPUTAPLEX es una plataforma científico-tecnológica del proyecto TIN 2011-23216 del Grupo de Investigación en Ingeniería de Software Empírica (GRISE) de la Universidad Politécnica de Madrid. Esta plataforma tecnológica tiene como objetivo facilitar a los investigadores la realización de tareas experimentales, replicación, síntesis y toma de decisiones basadas en eviden-

cias empíricas a cerca del proceso de desarrollo de software, además de permitir el intercambio, distribución y compartición de resultados experimentales entre grupos de investigación.

1.4. Descripción de objetivos del Proyecto SARIDIF

Los objetivos del proyecto tecnológico SARIDIF o Software de Agentes de Recuperación de Información de Diversas Fuentes se orientan a proveer nuevas funcionalidades a la plataforma COMPUTAPLEX, de tal manera que este pueda cumplir su plan de desarrollo previsto. Para lo cual, uno de los objetivos del presente TFM es realizar el estudio tecnológico arquitectónico de la plataforma COMPUTAPLEX, con el fin de comprender las tecnologías con las cuales se encuentra construido el software y los lineamientos que deben seguirse cumpliendo.

Un segundo objetivo de este TFM consiste en implementar componentes para la recuperación de datos experimentales disponibles en varias fuentes de datos para COMPUTAPLEX a partir de un mecanismo genérico y extensible que mantenga los lineamientos de los modelos conceptuales creados para la unificación de repositorios experimentales independientes.

El tercer objetivo se orienta a diseñar y hacer uso de una ontología de acciones para la comunicación entre nodos de la plataforma, para ello se establecerá la terminología correspondiente al dominio de la ISE, en la cual se considere únicamente las acciones que pueden atender los nodos de COMPUTAPLEX.

Finalmente, el último objetivo consiste en utilizar una metodología que permita incorporar las funcionalidades requeridas de la aplicación a medida que se tiene mayor conocimiento del proceso experimental en ingeniería de software.

1.5. Contribuciones

La primera contribución que SARIDIF plantea realizar es la construcción de componentes que permitan extraer la información experimental de las fuentes asociadas a cada nodo de la plataforma, para ello se hará uso de un mecanismo genérico que tiene la particularidad de no ser específico para cada una de las fuentes de datos, sino más bien permitirá asociar nuevas fuentes sin tener que preocuparse por refactorizar el código existente.

Otra contribución corresponde con el planteamiento de una estructura particular XML orientada a describir resultados experimentales que se encuentran registrados en hojas de cálculo y ficheros SPSS. Ello permitirá a esta infraestructura tecnológica utilizar fuentes de datos Excel y SPSS para la recuperación, unificación y consolidación de datos del proceso de experimentación en Ingeniería de Software.

Finalmente, el último aporte al proyecto corresponde con la definición e implementación de una ontología que será utilizada en la comunicación entre nodos de la Plataforma COMPUTAPLEX de tal manera que se mantenga una interacción entre los diferentes agentes haciendo uso de un vocabulario común con significado claro de conceptos de experimentación para efectuar las acciones de solicitud de información entre nodos.

1.6. Estructura de la memoria

La estructura de los capítulos de la presente memoria se divide de la siguiente manera:

El segundo capítulo describe los antecedentes del proyecto, en el que se describe el contexto en el que se desarrolla el proyecto COMPUTAPLEX, los objetivos de este proyecto, los principales usuarios, la arquitectura inicial y su plan de desarrollo previsto.

En el tercer capítulo se presenta el estado de la cuestión, se inicia con una descripción de la tecnología de agentes, los problemas que presenta esta tecnología en cuanto al acto comunicativo, seguidamente se analiza los orígenes de los lenguajes de comunicación junto con sus organismos fundadores, posteriormente se hace una descripción de FIPA para luego exponer una descripción de la plataforma JADE como implementación de FIPA y finalmente exponer el tema de Ontologías.

Luego, en el cuarto capítulo se presentan los objetivos del presente Trabajo de Fin de Máster, en el que constan tanto el objetivo general como los objetivos específicos, los cuales se encuentran alineados al proyecto científico tecnológico COMPUTAPLEX.

Seguidamente, en el quinto capítulo se explica la metodología empleada para la implementación de los componentes del proyecto COMPUTAPLEX. Se describe las etapas por las que atravesó el desarrollo del software, Además, se da a conocer cuál fue el plan de trabajo.

En el sexto capítulo se explica en detalle cada uno de los ciclos de desarrollo de acuerdo a las fases de la metodología ASD junto con sus correspondientes tareas realizadas y los resultados alcanzados tanto de los componentes creados para añadir características al proyecto COMPUTAPLEX como de la implementación de ontología.

Finalmente, para el séptimo capítulo se hace la discusión de la utilidad de construir los componentes para la recuperación de datos experimentales, frente al estado anterior de la plataforma COMPUTAPLEX, Además se describe las conclusiones a las que se ha llegado en el presente TFM.

Capítulo 2

Antecedentes

En este capítulo se da a conocer el contexto del proyecto COMPUTAPLEX. Se describe las etapas, objetivos y subobjetivos del proyecto, la arquitectura, los usuarios, y el plan de desarrollo previsto para implementar el proyecto.

2.1. Contexto del Proyecto COMPUTAPLEX

El proyecto COMPUTAPLEX es parte del proyecto científico TIN 2011-23216, tiene como objetivo apoyar a la comunidad científica de Ingeniería de Software Experimental la realización de las tareas de investigación sobre el desarrollo de software. COMPUTAPLEX es una infraestructura tecnológica de soporte para el proceso experimental. El soporte que proporciona al investigar en IS incluye la gestión de la información experimental, el flujo de información entre fuentes heterogéneas y la estandarización de la diversidad terminológica que se presenta en la comunidad de ESE. Esta plataforma tecnológica facilita a los investigadores compartir los resultados de sus investigaciones con otros grupos de investigación, con el fin de realizar análisis de resultados ya no solo de manera aislada, sino que se lo pueda hacer a gran escala concatenando conjuntos de experimentos.

COMPUTAPLEX posee la capacidad de gestionar ontologías dentro de la infraestructura, permitiendo a los nodos generar modelos de representación en RDF/XML. Por otro lado ofrece servicios web capaces de procesar dichas representaciones para ser expuestos y consumidos por otras

aplicaciones.

Otra de las funcionalidades que actualmente dispone COMPUTAPLEX es un browser de información experimental que permita dar a conocer y visualizar de forma sencilla qué información está disponible de cada uno de los experimentos que se han realizado dentro de un grupo o grupos de investigación, sin necesidad de conocer quién (experimentador) tiene la información, cómo está organizada o qué componentes tecnológicos están involucrados.

La planificación para la implementación del proyecto COMPUTAPLEX se la puede observar en la figura 2.1. Este plan está constituido por cuatro etapas, para cada una de estas se desagrega los objetivos y subobjetivos. Las etapas del plan corresponde con la secuencia del proceso experimental: experimentos, replicación, síntesis y tomas de decisiones.

2.2. Objetivos de COMPUTAPLEX

Los objetivos de COMPUTAPLEX están alineados con los objetivos del proyecto TIN 2011-23216. Los objetivos están orientados a cubrir las diferentes etapas del proceso de Experimentación en Ingeniería de Software. Este proceso está formado por cuatro etapas: Experimentación, Replicación, Síntesis y Toma de decisiones. Para cada uno de estas se ha planteado un objetivo general, del que se desagrega varios subobjetivos que corresponden a implementaciones de COMPUTAPLEX que pueden ser gestionadas de forma más efectiva, con lo cual el logro de cada uno de estos subobjetivos cubrirán dicho objetivo general. Las etapas y sus objetivos se describen a continuación:

La primera etapa denominada “Experimento” busca crear un Repositorio Experimental Centralizado como puede ser una base de datos, en donde se pueda consolidar la información de los

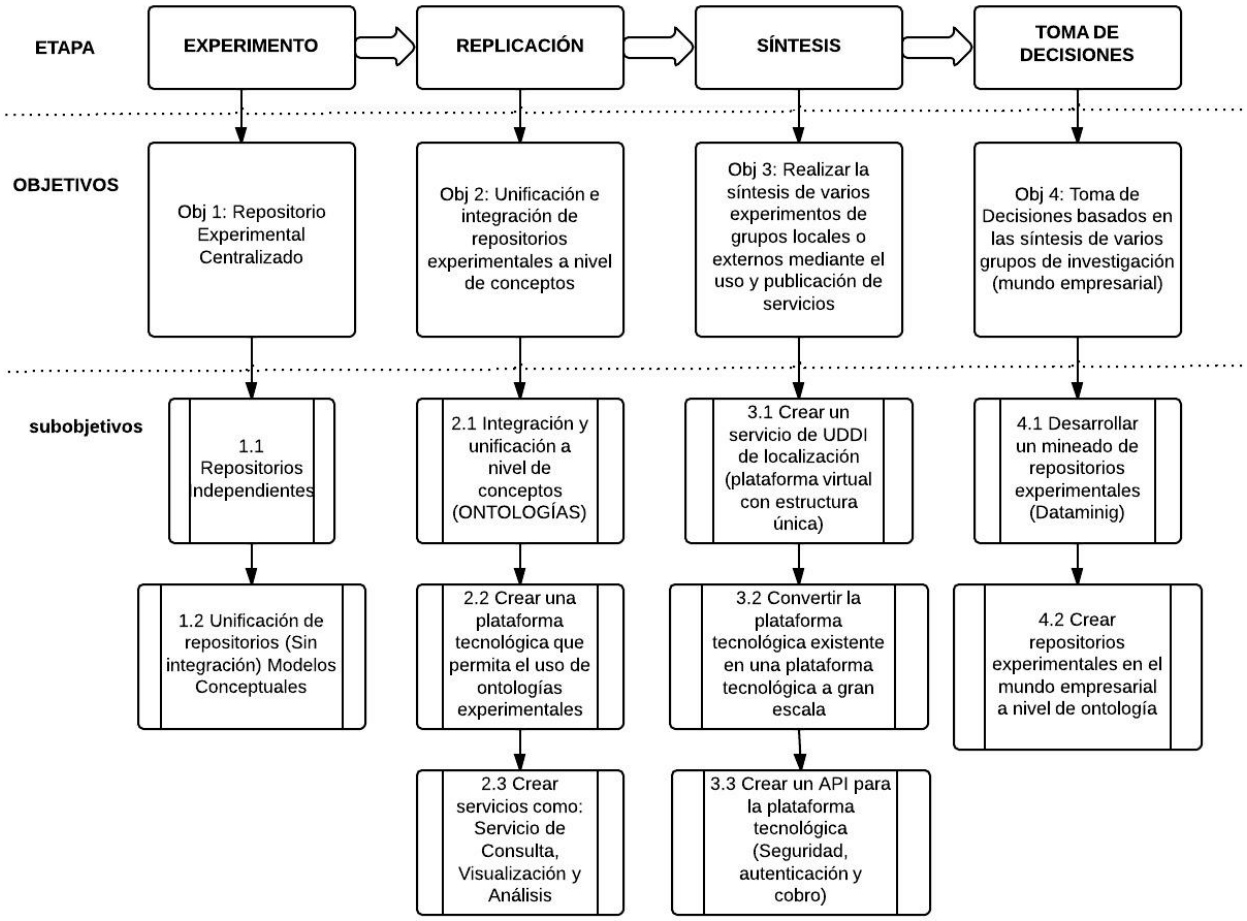

Figura 2.1: *Plan Desarrollo del Proyecto COMPUTAPLEX*

experimentos llevados a cabo por los investigadores, de manera que todos estos datos se puedan mantener en un único almacén centralizado. A partir de los repositorios independientes de los investigadores se busca crear un mecanismo de unificación, sin que se llegue a dar la integración entre estos. Dicha unificación se podrá realizar mediante el desarrollo de Modelos Conceptuales que permitan representar los datos comunes que mantienen los distintos repositorios independientes, para así tener una visión global y unificada de estos datos.

La segunda etapa corresponde con la "Replicación" la cual parte de la unificación de repositorios independientes establecida en la etapa anterior. Para esto es necesario no únicamente contar con el modelo conceptual global sino que ahora se define en un nivel de Ontologías, es decir a partir de un conjunto de conceptos relacionados con el dominio del proceso experimental. La unificación en este nivel será dado a partir de representaciones RDF/XML la cual oculta los repositorios independientes y provee del mecanismo para brindar a los investigadores la visión global de que están trabajando sobre un único repositorio. Para este fin es necesario desarrollar los servicios de consulta, visualización y análisis.

La tercera etapa corresponde a la "Síntesis" de varios experimentos tanto de grupos locales como externos, mediante el uso y publicación de servicios web que den acceso a los repositorios de los distintos grupos de investigación (eg. ESPEL - Ecuador, UPM - Madrid, UPV - Valencia). Para la publicación y localización de los diferentes servicios web será necesario crear un servicio UDDI (Universal Description, Discovery and Integration). Esto permitirá posicionar a COMPUTAPLEX como una estructura única de gran escala que permite la cooperación entre grupos de investigación, lo cual involucra dotar a dicha plataforma de ciertas características en cuanto a la seguridad, autenticación y monetización de COMPUTAPLEX, por lo que se ha previsto el desarrollo de APIs que provean estos aspectos.

La cuarta etapa corresponde a la "Toma de decisiones" en base a la información extraída en la

etapa de síntesis de varios experimentos, con lo que COMPUTAPLEX pasará a proveer servicios en el mundo empresarial, evaluando los repositorios experimentales mediante el uso de las técnicas de Datamining, para posteriormente pasar a una etapa de creación de repositorios experimentales a nivel ontológico en el que se integre la información experimental del campo académico con los del mundo empresarial.

2.3. Estado actual del proyecto COMPUTAPLEX

La plataforma COMPUTAPLEX se encuentra actualmente en la etapa de Replicación, dispone de funcionalidades y capacidades necesarias para la unificación de repositorios a nivel de modelos conceptuales, su desarrollo se lo ha hecho bajo la tecnología de agentes mediante el framework JADE, en relación con el mecanismo de unificación de repositorios hace uso de un modelo basado en ontologías mediante el framework JENA, de tal manera que la recuperación de datos de los diferentes repositorios estén incorporados en una única vista global para los investigadores. Además, dicha plataforma consta de componentes capaces de brindar servicios en cuanto a la consulta, visualización y análisis de resultados, permitiendo que los datos puedan ser transmitidos por diferentes plataformas tecnológicas para lo cual se requiere que exista interoperabilidad entre sistemas.

Los participantes del proyecto COMPUTAPLEX son el Grupo de Investigación en Ingeniería de Software Empírica - GRISE de la Universidad Politécnica de Madrid, quienes cumplen con ciertos roles y funciones para el proyecto. En el cuadro 2.1 se puede apreciar la estructura organizacional, en la que se establece los niveles jerárquicos para llevar a cabo las actividades de coordinación y comunicación para lograr una adecuada gestión del proyecto.

2.4. Arquitectura Inicial del Proyecto

Antes de iniciar la implementación del presente proyecto de TFM, es necesario conocer la arquitectura con la cual el proyecto COMPUTAPLEX se encuentra desarrollada, así como tam-

Fuentes:	Grupo de Investigación de Ingeniería de Software Empírica- Universidad Politécnica de Madrid
Interesados:	Investigadores COMPUTAPLEX
Líder del Proyecto:	Dña. Natalia Juristo
Individuos del Grupo:	D. Oscar Dieste, D. Rodrigo Fonseca, D. John Castro, D. Oscar Testa
Usuarios COMPUTAPLEX:	Dña. Natalia Juristo, D. Oscar Dieste, Dña. Sira Vegas

Cuadro 2.1: *Participantes del proyecto COMPUTAPLEX*

bién su estado actual, para a partir de allí poder agregar nuevos componentes que provean nuevas funcionalidades a la plataforma. En la figura 2.2 se muestra la arquitectura de COMPUTAPLEX.

El proyecto COMPUTAPLEX está desarrollado bajo la tecnología de sistemas multiagentes, bajo el framework de implementación JADE (Java Agent Development). Consta de dos roles de agentes, el primero de ellos es el cliente el cual es el encargado de hacer las peticiones de información experimental hacia los agentes publicadores, los cuales cumplen la tarea de extraer información experimental de los repositorios que están asociados a estos (eg. Base de Datos). Una de las funciones que cumple el agente publicador es representar la información experimental mediante un modelo a nivel conceptual utilizando ontologías.

En cuanto a la transmisión de mensajes entre agentes se lo realiza mediante un agente provisto por la plataforma JADE, denominado Directory Facilitator(DF), el cual tiene conocimiento de todos los agentes clientes y publicadores registrados hasta ese momento en dicha plataforma.

Para realizar dicha representación de los repositorios experimentales se utiliza un esquema RDF/XML, el cual es generado dinámicamente a partir de los datos extraídos de la fuente de datos con la que el agente mantiene su enlace. La generación de los esquemas se lo realiza en JENA un framework Open Source para la creación de aplicaciones linked data y web semántica.

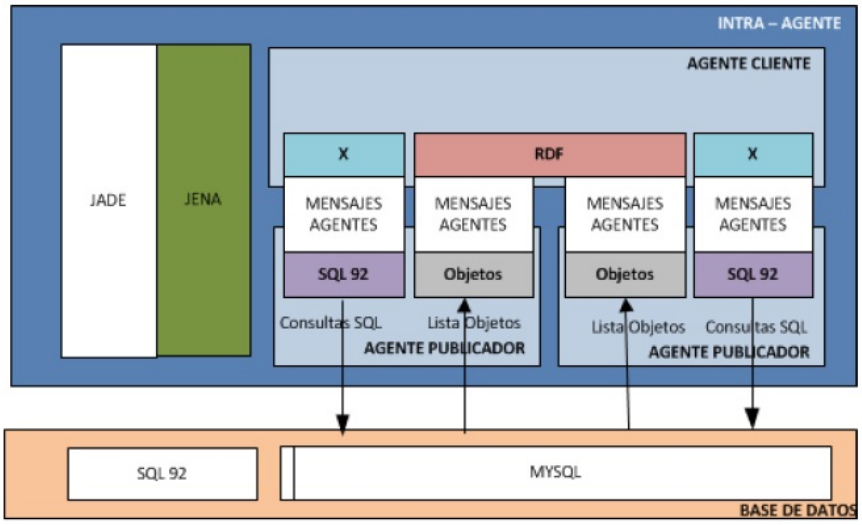

Figura 2.2: *Arquitectura del Proyecto COMPUTAPLEX*

En cuanto a la secuencia de interacción que se produce en relación al acto comunicativo entre agentes de la plataforma COMPUTAPLEX para realizar la extracción de datos se puede describir diciendo que esta actividad la inicia el agente cliente emitiendo una solicitud al Directory Facilitator para que éste informe de los agentes publicadores que se encuentran registrados y puedan atender dicha petición. Una vez que esta información es devuelta al agente cliente, este empieza a emitir a cada agente publicador una solicitud para que extraiga la información de los experimentos desde los repositorios de datos experimentales que se encuentran asociados, para a continuación crear de manera dinámica la representación ontológica en RDF/XML que serán devueltos como respuesta al agente cliente, para que este a su vez los unifique proporcionando a los investigadores la observación de un único almacén virtual.

Capítulo 3

Estado de la cuestión

En el presente capítulo se pretende dar una visión de la tecnología de agentes, sus características, los problemas de comunicación, los lenguajes de comunicación, la descripción FIPA, la plataforma JADE como implementación FIPA y finalmente una descripción de las Ontologías.

3.1. Definiciones de Agentes

Hasta la actualidad no existe una definición universalmente aceptada sobre el término agente computacional. Pese a que existe gran cantidad de definiciones dadas por varios investigadores, estas están hechas desde las perspectivas del campo de investigación en el cual trabajan los investigadores.

Una de las definiciones más utilizadas es la de Wooldridge y Jennings la cual dice un agente: "Es un programa autocontenido capaz de controlar su proceso de toma de decisiones y de actuar, basado en la percepción de su ambiente, en persecución de uno o varios objetivos".

Por otro lado, se habla de la existencia de dos nociones sobre agentes:
La noción débil de agente considera a un agente como una entidad capaz de intercambiar mensajes utilizando un lenguaje de comunicación, se caracteriza por ser autónomo, comunicativo, reactivo, proactivo, temporalmente continuo[32], puesto que su fin es lograr la interoperabilidad entre aplicaciones en un nivel semántico.

La noción fuerte o restrictiva enuncia que un agente es inteligente si incorpora: nociones de conocimientos, deseos, intensiones, aprendizaje, carácter, movilidad, veracidad o benevolencia[2]. Como agentes inteligentes realizan continuamente tres funciones: perciben las condiciones dinámicas del entorno, actúan para influir en estas condiciones del entorno, y razonan para interpretar percepciones, resolver problemas, definir inferencias y determinar acciones[11].

Ferber[8] caracteriza al agente software por ser un sistema computacional abierto (ensamble de aplicaciones, redes y sistemas heterogéneos) que puede comunicarse con otros agentes, que es guiado por sus propios objetivos y recursos, mantiene una representación parcial de otros agentes, posee servicios que puede ofrecer a otros agentes, su comportamiento se orienta a cumplir con sus objetivos, tomando en cuenta sus recursos y habilidades disponibles y dependiendo de sus representaciones y las comunicaciones que recibe. La figura 3.1 muestra gráficamente la representación de un agente.

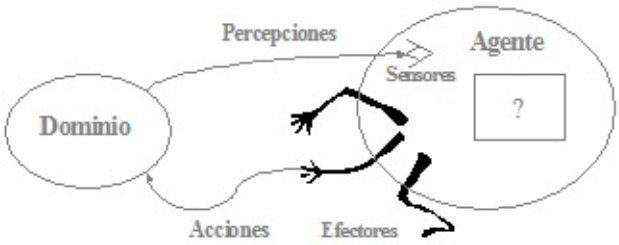

Figura 3.1: *Concepto de Agente*

Todos los agentes inteligentes son programas, pero no todos los programas que realizan búsquedas son agentes inteligentes. Así tampoco un agente puede ser llamado inteligente si este no es capaz de adaptarse y realizar bien sus funciones cuando se ubica en un entorno diferente del cual fue diseñado[1]. Los agentes por sí mismos pueden considerarse entes individuales que se comunican con otros agentes para resolver de forma adecuada su trabajo. Son también considera-

dos procesos que se encuentran ejecutándose continuamente; saben qué hacer y cómo hacer para alcanzar sus objetivos.

3.2. Sistemas Multi-agente

Es conveniente considerar que existe una diferencia terminológica entre los sistemas basados en agentes y los sistemas multiagentes. Un sistema basado en agentes es aquel que utiliza el concepto de agente como mecanismo de abstracción, pero aunque sea modelado en términos de agentes podría ser implementado sin ninguna estructura de software correspondiente a éstos. Por otro lado los sistema multiagentes son diseñados e implementados pensando en que estarán compuestos por varios agentes que interactuan entre sí, de forma que trabajen juntos para alcanzar los objetivos comunes. Por lo que es necesario hacer un mayor esfuerzo de abstracción, identificar mecanismos de aprendizaje, coordinación, negociación entre otros[14].

Los agentes de software pueden ser más útiles cuando trabajan en conjunto con otros agentes en la realización de tareas. Por lo que la programación de sistemas multiagentes permite al desarrollador implementar el sistema usando múltiples agentes, especializando a cada uno de estos en un conjunto de tareas particulares. Todos estos agentes necesitan comunicarse con los demás y deben tener la capacidad de trabajar juntos para alcanzar sus objetivos comunes.

Un sistema multiagente (SMA) puede concebirse como un conjunto de entidades autónomas o sociedad de agentes, donde las interacciones entre agentes y su entorno proporcionan un comportamiento satisfactorio de todo el sistema[2]. También puede ser definido como una red débilmente acoplada de agentes resolvedores de problemas complejos que interactúan entre sí, aportando sus capacidades y conocimientos individuales, puesto que el programador ha dotado al agente de las capacidades necesarias para actuar en determinadas situaciones[29]. Aún cuando suelen ser de naturaleza heterogénea estos tienen la facilidad para cooperar y coordinar tareas en forma distribuida.

3.3. Problema de comunicación entre agentes

La tecnología de agentes en sus inicios usó soluciones propias para el acto comunicativo, pues la necesidad de interacción y comunicación, es parte fundamental de un sistema multiagente. Sin embargo, un problema habitual sucede cuando los agentes son desarrollados de manera independiente, lo que conlleva a un problema en cuanto a la falta de comunicación[9].

Uno de los problemas que ocurren con la tecnología de agentes es que pueden perder la comunicación debido a nodos que se encuentran inactivos, aun cuando aparenten estar interactuando sin lograr en realidad comunicarse, esto puede darse debido a: fallas en el hardware, la imposibilidad de comunicación o por excesiva autonomía que poseen los agentes[9].

Otro problema se da con los mecanismos de negociación por la escasez de recursos, causando que no todos los objetivos se puedan cumplir, generalmente esto ocurre cuando existe escasez de recursos y varios agentes entran en conflicto, pues al existir limitaciones de coordinación y sincronización en la provisión de servicios, los recursos no podrán suplir todas las demandas de los agentes. Para este tipo de problemas la solución consiste en permitir una negociación interactiva en la oferta y la asignación de tareas. Para este fin cada nodo indica a qué tareas puede comprometerse a realizar y a cuáles no, de modo que los nodos puedan obtener posibles soluciones globales, sin tener una visión de la solución global.

Otro tipo de problemas se presenta cuando se detecta un conflicto entre agentes, para ello se emplea la centralización de tareas, que consiste en elegir a un agente para que resuelva el conflicto. Este agente elegido tiene que replanificar el sistema, retransmitir este plan y ejecutarlo.

Existen también problemas en cuanto a los denominados actos del habla o actos comunicativos, que se implementan dentro de los agentes pues no solo es necesario entender el significado de cada frase para comprender un texto, sino también se requiere comprender las intenciones de los interlocutores y cómo se desarrolla el diálogo pues no toda comunicación es explicita[9]. Para dar

solución a este tipo de problemas se han generado varias iniciativas, que han tratado de especificar una serie de estándares que consigan una interacción real y homogénea entre diversos sistemas multiagentes.

3.4. Primeros lenguajes de comunicación entre agentes

Entender la evolución del concepto ACL (Agent Communication Language) requiere entender el contexto en que nació KQML (Knowledge Query and Manipulation Language)[18]. KQML fue por primera vez introducido como un resultado del KSE (Knowledge Sharing Effort). KQML usa una sintaxis similar a ACL[21] y tiene sus inicios en un programa de investigación aproximadamente en los años 1990, fue impulsada por la agencia DARPA (Defense Advanced Research Projects Agency) del Gobierno de los Estados Unidos junto con la participación tanto de la academia como de la industria[19].

En el modelo KSE, los agentes (sistemas basados en conocimiento) son vistos como bases de conocimiento virtual[18], que intercambian proposiciones usando un lenguaje que expresa varias actitudes proposicionales, las cuales son relaciones conformadas por tres partes[19]:

- Un agente.
- Un contenido de soporte proposicional (e.g está lloviendo).
- Un conjunto finito de actitudes proposicionales que un agente podría tener con respecto a otra proposición (e.g creencias, aserciones, miedos, entre otros).

Un ejemplo en la que se observa la relación entre las partes de una actitud proposicional se muestra a continuación:

<agenteA,miedo,llueve(ahora)>.

El modelo KSE incluye tres capas de representación: 1) Especificación de actitudes proposicionales. 2) Especificación de proposiciones (Conocimientos). 3) Especificación de una ontología (vocabulario). Adicionalmente, el modelo incluye un componente de lenguaje para las actitudes proposicionales, denominado lenguaje para manipulación y consulta de conocimiento (KQML) y un formato de intercambio de conocimiento (KIF).

Lenguaje de manipulación y consulta de conocimiento

KQML es conocido como el lenguaje para manipulación y consulta de conocimiento, el cual es un protocolo de intercambio de información tanto para agentes como para programas de aplicación. Se trata de un lenguaje de comunicación de alto nivel orientado a mensajes que es independiente del protocolo de transporte (TCP/IP, SMTP entre otros). Otra característica de KQML es ser independiente del lenguaje de codificación del contenido del mensaje (e.g KIF, SQL, PROLOG, DELP, entre otros), así como también de la ontología empleada para el contenido[19].

KQML incluye muchas primitivas, las cuales son directivas para que los agentes puedan comunicar, responder cuestiones, suscribir servicios o encontrar a otros agentes. Su objetivo fue desarrollar las técnicas, metodologías y herramientas de software para compartir y reusar el conocimiento entre los sistemas software. Además, de definir un lenguaje común para el proceso de compartición de conocimiento entre sistemas multiagentes.

Yannis Labrou señala que a nivel conceptual, un mensaje KQML tiene tres niveles[19]: el de contenido, de mensaje y comunicación. Estos niveles pueden apreciarse en la siguiente figura 3.2.

El nivel de contenido: está relacionado con el contenido real del mensaje a ser transmitido, es codificado en algún lenguaje de representación elegido por cada agente (Pueden usar lenguajes expresados como cadenas ASCII o de notación binaria). KQML puede llevar cualquier codificación en este nivel, pues en su definición dice que toda implementación debe ignorar dichos aspectos,

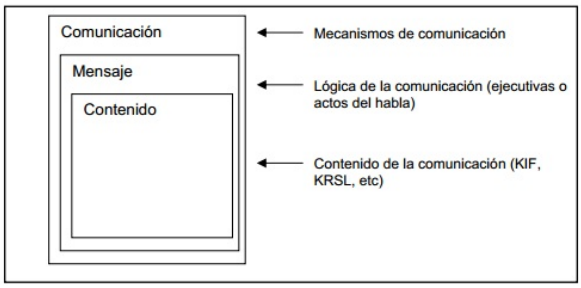

Figura 3.2: *Estructura del lenguaje KQML*

esto es que ignora la parte del contenido exceptuando la magnitud que se utiliza para conocer donde termina.

El nivel de mensaje: codifica un mensaje que un agente desea transmitir a otros, representa el núcleo del lenguaje KQML. Este nivel determina el protocolo de red con el que se enviará el mensaje, incluye el acto de habla o performativa que el transmisor adjunta al contenido, este acto de habla indica si el mensaje es una aserción, una pregunta, comando o cualquier otra performativa válida. Además en esta capa se incluyen características opcionales que describen el contenido: su lenguaje y la ontología que se usa. Esto permite que las implementaciones KQML realicen un análisis, ruteo y entrega del mensaje apropiadamente aunque su contenido permanezca inaccesible.

El nivel de comunicación: codifica un conjunto de características para el mensaje, describen los parámetros de la comunicación a un bajo nivel tales como la identidad del emisor y del receptor. Esta capa agrega una segunda capa de características como la identidad, transmisor y receptor, identificador único de comunicación y si el tipo de comunicación es síncrona o asíncrona.

En la figura 3.3 se muestra un ejemplo de un mensaje KQML que un agente llamado joe envía a un servidor para hacer una consulta sobre el precio del inventario de IBM. En este ejemplo la performativa KQML es ask-one, el contenido es (PRICE IBM ?price), la ontología es identificada por el token NYSE-TICKS, el receptor del mensaje es stock-server y la consulta está escrita

en lenguaje LPROLOG. EL valor de :content corresponde a la capa de contenido; los valores de :reply-with, ;sender y receiver corresponden a la capa de comunicación y, finalmente el nombre de la perfomative, :language y :ontology forman la capa de mensaje.

```
(ask-one
        :sender joe
        :content (PRICE IBM ?price)
        receiver stock-server
        :reply-with ibm-stock
        :language LPROLOG
        :ontology NYSE-TICKS)
```

Figura 3.3: *Mensaje KQML de consulta*

Un mensaje KQML de respuesta enviada por el servidor llamado stock-server, se puede apreciar en la figura 3.4.

```
(tell
        :sender stock-server
        :content (PRICE IBM 14)
        receiver joe
        :in-reply-to ibm-stock
        :language LPROLOG
        :ontology NYSE-TICKS)
```

Figura 3.4: *Mensaje KQML de respuesta*

KQML como ya se mencionó usa un conjunto de actos comunicativos denominadas performativas. Este conjunto está abierto a que nuevas performativas puedan ser agregadas. Dichas performativas están clasificadas en tres categorías:

- **De discurso.** Se utilizan en el contexto de interacciones orientadas al intercambio de información y conocimiento entre dos agentes.

- **De intervención y mecánica de la conversación.** Se utilizan para intervenir en la conversación de agentes o bien para terminarla prematuramente o ignorar el protocolo de comunicación por defecto.

- **De comunicación en red y facilitación.** Permiten a los agentes encontrar a otros agentes que puedan dar respuesta a sus preguntas. Esta búsqueda se hace a través de los agentes facilitadores, los cuales están capacitados para transmitir información entre toda la comunidad.

Formato de intercambio de conocimiento

KIF es una solución sugerida por el KSE diseñada para el intercambio de conocimiento entre bases de conocimiento heterogéneas, provee los aspectos sintácticos para la representación de conocimiento compartido, es un lenguaje propuesto como un vehículo para expresar conocimiento y meta-conocimiento[24], está basado en la lógica de predicados, permite cierta flexibilidad en el lenguaje de representación al soportar la definición de objetos, funciones, relaciones, reglas y metaconocimiento. KIF es una representación interna del conocimiento, y en cuanto sea posible realizar la lectura y escritura por agentes software, este conocimiento será portable y reusable; por ende podrá ser compartido por distintos agentes y estar distribuido por el sistema[20].

Otro de los lenguajes ampliamente utilizado es FIPA ACL, el cual al igual que KQML están diseñados para proporcionar una base linguistica común para que agentes autónomos puedan comunicarse entre sí. Ambos lenguajes tienen una sintaxis similar que está basado en el lenguaje LISP, incluyendo la forma y los parámetros de los mensajes.[2].

3.5. Descripción FIPA

FIPA (Foundation for Intelligent Physical Agents) es un consorcio industrial fundado en 1996 por empresas y organizaciones del área de telecomunicaciones e informática para promover el desarrollo de especificaciones internacionales para tecnologías de agentes. Estas especificacio-

nes describen diferentes características relacionadas con los agentes como son: la gestión de los mismos, el lenguaje de comunicación, interacción agente-humano, movilidad, seguridad, representación de los mensajes.

La organización FIPA está conformada por tres comités cada uno de los cuales tiene su función específica. El comité técnico es el responsable de producir, mantener y actualizar la especificación. El comité de gestión cubre los aspectos relacionados con los servicios de los agentes para la provisión, registro de la plataforma de agentes. Finalmente el comité de interacción de agentes cubre la integración de los agentes en los sistemas software. El objetivo FIPA es hacer disponible la especificación que maximice interoperabilidad entre sistemas basados en agentes. En cuanto a la definición de la plataforma de agentes, FIPA sigue el principio de definir únicamente el comportamiento externo (su interfaz), dejando las decisiones de diseño a cada equipo de desarrollo de esta tecnología[7].

Un principio que guía el estándar es el conseguir un sistema que sea totalmente abierto para que distintos sistemas heterogéneos puedan interactuar a nivel de sociedades de agentes. Además, se establece un modelo lógico capaz de gestionar la creación, destrucción registro, localización y comunicación de agentes[7].

FIPA proporciona la plataforma de agentes, en la que se definen los servicios relacionados con el transporte de mensajes, un sistema de gestión de agentes, un servicio de directorio y canal de comunicación. Cada uno de estos servicios excepto el de transporte de mensajes, es suministrado por agentes especializados, dando lugar a una comunicación basada en mensajes ACL. Estos servicios se pueden apreciar en la figura 3.5.

Al igual que KQML, FIPA-ACL fue definido con la teoría de los actos del habla, mantiene una sintaxis similar salvo ciertas primitivas que han sido denominadas de diferente forma. Adopta

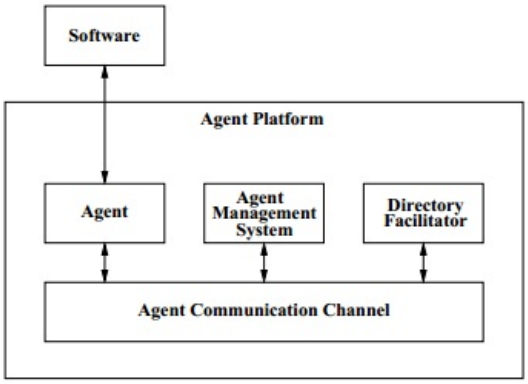

Figura 3.5: *Arquitectura Abstracta FIPA*

la política de separación del lenguaje interno de los mensajes de su lenguaje externo. El lenguaje externo denota el significado pretendido del mensaje, mientras que el lenguaje interno representa el contenido al cual se aplican los deseos, intenciones o creencias del interlocutor[28].

Los mensajes entre agentes se interpretan como acciones o como actos comunicativos, pues al ser enviados supone un agente realizará una acción particular. La especificación consiste en un conjunto de tipos de mensajes junto con las actitudes mentales que tiene el emisor y receptor.

Los proyectos de software multiagentes distribuyen las entidades de los sistemas en entornos abiertos, estos sistemas son más complejos de desarrollar que los sistemas tradicionales. En los sistemas abiertos las entidades cambian continuamente puesto que hay entidades que aparecen, se modifican, se destruyen y actúan concurrentemente, agregando además la complejidad del trabajo con grandes volúmenes de información y variación de comportamientos de los agentes, pudiendo existir actividades tanto de cooperación como de competición por los recursos. Los agentes pueden detectar si una tarea se la está realizando de la manera planificada o pueden decidir cambiarla por otra[21].

Otros aspectos a considerar es la capacidad de procesamiento y gestión del conocimiento, con

el uso de ontologías y lenguajes de comunicación de agentes existe ventaja frente a la comunicación habitual puesto que los objetos distribuidos tradicionales se limitan a la interoperabilidad sintáctica.

3.6. Descripción de JADE como implementación de FIPA

La plataforma JADE[1] (Java Agent DEvelopment Framework) fue desarrollada por Telecom Italia (CSELT) en 1998, es distribuida bajo la licencia LGPL[2] (Library Gnu Public License)[4]. JADE es considerado un middleware[32] para el desarrollo de sistemas multiagente distribuidos que respeta la especificación FIPA, su arquitectura de comunicación es peer-to peer. Este framework, permite desarrollar agentes móviles software específicos para satisfacer las necesidades del usuario. Además del framework, JADE ofrece una plataforma donde estos podrán ser ejecutados[27].

En la plataforma JADE los peer o agentes pueden agregarse o eliminarse dentro del entorno según las necesidades del sistema, la comunicación entre agentes se puede realizar en diferentes tipos de redes de datos, como son las redes tradicionales o redes inalámbricas, sin distinguirse entre servidores y clientes. JADE es totalmente desarrollado en Java[4] y posee los siguientes principios:

- Interoperabilidad: JADE sigue las especificaciones FIPA que proporcionarán un estándar para que los agentes puedan cooperar y colaborar en la consecución de objetivos.
- Uniformidad y Portabilidad: JADE proporciona un conjunto de API's que son independientes de la red y de la tecnología Java (J2SE, J2EE, J2ME) subyacentes.
- Fácil de Usar: La complejidad del middleware es abstraída por el conjunto de API's cuyo uso es muy intuitivo para el programador. Incluso ocultando la complejidad en redes inalámbricas.

[1]"http://jade.tilab.com/"

[2]"La licencia LGPL asegura todos los derechos básicos para facilitar el uso del software incluido en los productos comerciales: el derecho a hacer copias del software y la distribución de esas copias, el derecho a tener acceso al código fuente, y el derecho a cambiar el código y hacer mejoras en él."

- Código abierto: JADE es un proyecto de código abierto permitiendo la colaboración de la comunidad de usuario.

Funcionalidades que JADE proporciona

Para Bellifemine entre las funcionalidades que JADE proporciona a los programadores para ser utilizadas o extendidas[4], se puede mencionar:

- Posee un sistema distribuido habitado por agentes, cada uno se ejecuta como un subproceso independiente, por lo general en diferentes máquinas remotas, y capaces de comunicarse entre sí de manera transparente, es decir, la plataforma proporciona una API independiente de la ubicación que abstrae la infraestructura de comunicación para este fin.

- Transporte eficiente de mensajes asíncronos. La plataforma selecciona los mejores medios de comunicación disponibles y, cuando es posible, evita la serialización y deserialización de objetos. Cuando un mensaje cruza los límites de la plataforma, los mensajes se transforman automáticamente de la representación java propia de JADE hacia la sintaxis FIPA.

- Gestión sencilla y eficaz del ciclo de vida del agente. Cuando se crean los agentes se les asigna automáticamente un identificador único global y una dirección de transporte que se utiliza para registrarse en el servicio de páginas blancas de la plataforma. Además tanto APIs y herramientas gráficas se suministran a nivel local para administrar de forma remota los ciclos de vida de los agentes, es decir, crear, suspender, reanudar, suspender, reanudar, migrar, clonar y eliminar agentes.

- Apoyo para la movilidad del agente. Los agentes pueden migrar entre procesos y máquinas bajo ciertas circunstancias, sin que se vea afectado la interacción con otros agentes incluso en la etapa de migración.

- Mecanismo de suscripción de los agentes, y aplicaciones, permiten que en una plataforma los agentes sean notificados de todos los eventos del ciclo de vida y del intercambio de

mensajes.

- Conjunto de herramientas gráficas de apoyo a los programadores en la etapa de depuración y mantenimiento. Estas funciones son particularmente importantes para el trabajo con aplicaciones multiprocesos y multihilos, se las lleva a cabo mediante sniffers y simuladores que pueden controlar y depurar paso a paso la ejecución remota de los agentes.

- Soporte para ontologías y lenguajes de contenido. La plataforma realiza verificación ontológica y codificación del contenido automáticamente, brindando la posibilidad a los programadores de seleccionar sus lenguajes de contenido y ontologías como por ejemplo XML y extensiones basadas en RDF, así como también implementar sus propios lenguajes de contendidos en base a sus requerimientos específicos de la aplicación.

- Biblioteca de protocolos de interacción. Proporcionan un modelo de patrones de comunicación orientados hacia la consecución de uno o más objetivos. La arquitectura de aplicación es independiente y dispone de un conjunto de clases java que se pueden personalizar. Los protocolos de interacción también se pueden representar e implementar como un conjunto de máquinas de estados finitos concurrentes.

- Integración con diversas tecnologías basadas en la Web. incluyendo JSP, servlets, applets y tecnología de servicios Web. La plataforma también puede ser configurada para atravesar firewalls y utilizar sistemas NAT.

- Soporte para la plataforma J2ME y entorno inalámbrico. El framework JADE está disponible en las plataformas J2ME CDC y CLDC a través de un conjunto uniforme de APIs que cubren tanto J2ME y entornos J2SE.

- Un kernel extensible que permite a los programadores extender funcionalidad de la plataforma a través de la adición de servicios distribuidos a nivel del núcleo.

La plataforma JADE está conformada por varios contenedores de agentes, que se encuentran distribuidos por la red de ordenadores y son quienes proporcionan los servicios necesarios para

la ejecución de los agentes, a este entorno donde habitan los agentes se lo denomina JADE Run-Time. El conjunto de todos los contenedores es una capa homogénea que oculta la complejidad y la diversidad de las capas subyacentes. (Hardware, SO, Tipo de Red, Máquina Virtual de Java, etc.)[15].

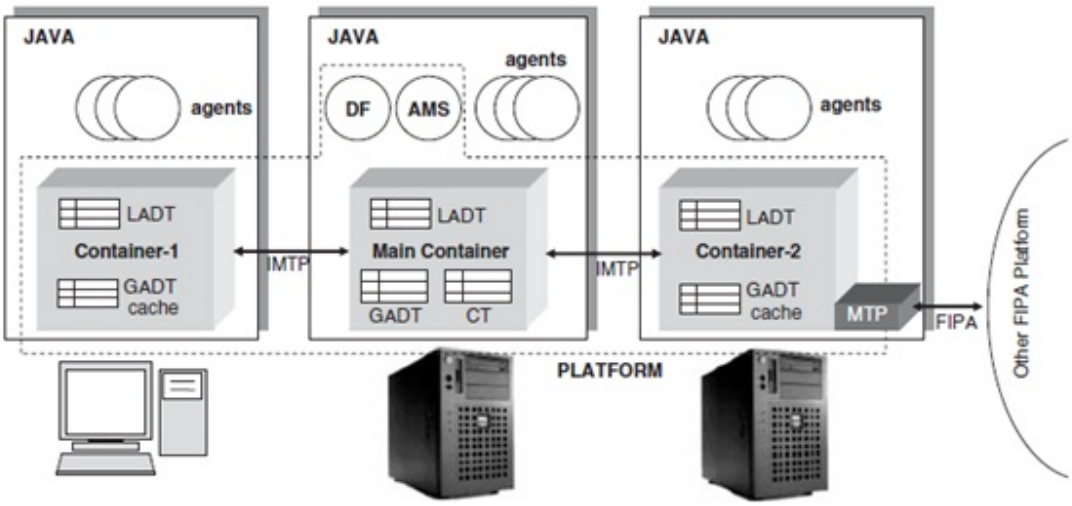

Figura 3.6: *Componentes de la plataforma JADE*

En la Figura 3.6 se puede observar los elementos de la arquitectura JADE, en la cual existe un Contenedor Principal que representa el punto de inicio de la plataforma: es el primer recipiente que se ejecuta y al cual todos los otros recipientes deben unirse registrándose con este. Cuando el contenedor-principal es iniciado, dos agentes especiales son automaticamente instanciados y lazados por JADE:

1. El Agent Management System (AMS) es un agente que supervisa la plataforma. Es el punto donde se realiza la supervición del acceso y uso de la plataforma, el AMS proporciona un servició de páginas blancas y del ciclo de vida, manteniendo una lista de los identificadores de los agentes (AID) y el estado en el que están.

2. El Directory Facilitator (DF) es un agente que proporciona los servicios de directorio de páginas amarillas, es usado para que cualquier agente que desee pueda registrar sus servicios

o buscar otros servicios disponibles. El DF también permite realizar suscripción de agentes que deseen ser notificados cuando se realiza un registro o modificación que concuerde con los criterios especificados.

El contenedor principal es el encargado de realizar la gestión de la tabla de contenedores, registrando las referencias a objetos y las direcciones de transporte de todos los nodos que componen la plataforma de contenedores. Una segunda tarea corresponde con la gestión de la tabla global de descripción de los agentes (GADT), en la cual se registran todos los agentes presentes en la plataforma, así como también su estado y la ubicación de cada agente. Finalmente otra tarea corresponde al alojamiento de la AMS y DF, proporcionando el servicio de páginas blancas y el servicio de páginas amarillas por defecto de la plataforma.

La figura 3.7 diagrama UML esquematiza la relación existente entre los diferentes elementos de la plataforma JADE. En el mismo se puede apreciar que una plataforma está compuesta por uno o varios Contenedores, el Main Container es un tipo de contenedor el cual se encuentra representado con el símbolo de la herencia, y en un contenedor pueden vivir los agentes.

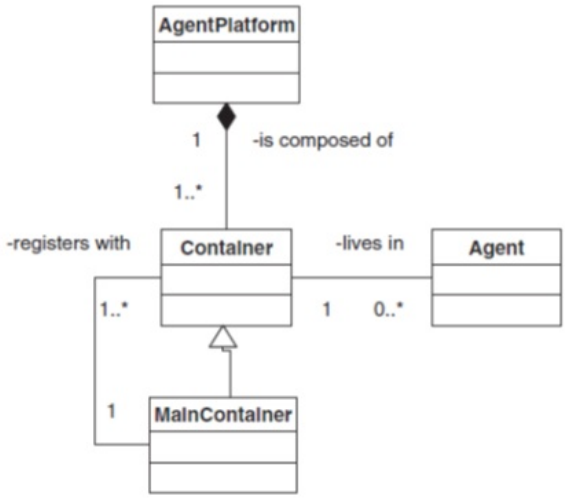

Figura 3.7: *Diagrama UML de elementos de la plataforma JADE*

3.7. Descripción de Ontología

El modelo de comunicación de FIPA entre agentes hace uso de mensajes ACL (Agent Comunication Language). Los mensajes ACL consisten en una expresión de un determinado lenguaje y con un conjunto de términos específicos de la ontología usada. Por lo que debe existir un acuerdo mutuo tanto en el lenguaje y la ontología empleada. Este modelo se aprecia en la figura 3.8.

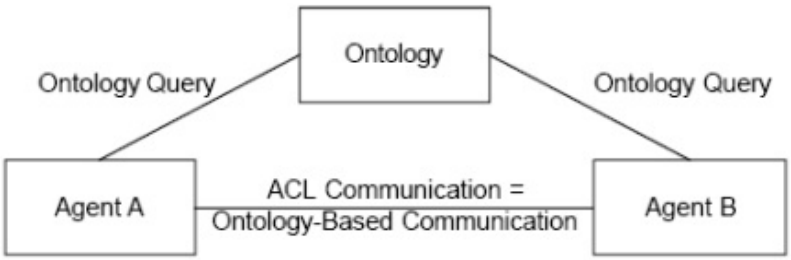

Figura 3.8: *Modelo de comunicación basado en Ontologías*

Las ontologías pueden definirse como un conjunto de símbolos o términos junto con su correspondiente interpretación o significado. En el ámbito de la comunicación entre agentes, si estos usan una misma ontología se asegura que lo que se está diciendo coincide con el significado de los términos utilizados en el mensaje.

Para Gruber en[10] una ontología es una especificación explicita de una conceptualización. Cuando el conocimiento de un dominio es representado en un formalismo declarativo, el conjunto de objetos que pueden ser representados es llamado el universo del discurso. Este conjunto de objetos y relaciones entre ellos, se reflejan en un vocabulario significativo con el cual un programa basado en conocimiento representa el conocimiento.

Cuando un agente se registra en el DF (Directory Facilitator) informa de las ontologías de las que tiene conocimiento, para así, cuando otro agente desee conversar con él utiliza una de ellas de

tal manera que se logre un entendimiento mutuo. Cuando el dominio es pequeño es posible que las ontologías se incluyan dentro del propio código del agente, cuanto esto sucede se denomina ontologías cerradas y limitadas al contexto en el que van a ser utilizadas.

En otro caso, cuando el sistema es de mayor envergadura se deben utilizar servicios de ontología (Ontology Agent), de tal forma que todos los agentes puedan utilizar dichos servicios para obtener toda la información necesaria. Entre las actividades que cumple un servicio de ontologías se tiene:

- Mantiene un conjunto de ontologías de uso público accesible a los agentes.
- Traduce expresiones entre diferentes ontologías.
- Responde a consultas sobre términos de las ontologías que gestiona.
- Facilita la identificación y uso de ontologías compatidas entre los agentes.
- Descubre nuevas ontologías y las pone a disposición de todos los agentes.

En la plataforma JADE una ontología es una instancia de la clase JADE.content.onto.Ontology en la cual se definen los esquemas de la estructura de los predicados (expresiones que relacionan conceptos), acciones de los agentes y conceptos (entidades) relevantes del dominio del problema.

3.8. Parser XML

Los parser o procesadores XML (eXtensible Markup Language) son software analizadores sintácticos de documentos XML, los cuales proveen las interfaces para descomponer un documento XML en sus diferentes elementos individuales, verificando que su estructura esté bien formada y que sea válida. Básicamente existe dos categorías principales de parsers XML: Document Object Model (DOM) y Simple API for XML (SAX).

DOM es una API (Application Program Interface) independiente del lenguaje para el acceso y modificación de documentos HTML o XML. DOM posee dos desventajas principales. La primera, el documento XML es almacenado completamente en memoria, por lo que puede afectar el rendimiento de la aplicación en el caso de que el documento sea demasiado grande. La segunda desventaja enuncia que al ser DOM una API genérica independiente del lenguaje, el procesamiento de un documento XML demanda mayor número de instrucciones comparado con una API específica para un tipo de lenguaje.

SAX es un parser basado en eventos que provee los mecanismos únicamente para lectura del contenido de un documento XML. Es un parser que opera en base a eventos mediante handlers que deben ser programados por el desarrollador según el tratamiento requerido, estos eventos son invocados a medida que el documento XML es procesado. La principal ventaja de SAX sobre DOM es que no requiere que el documento se encuentre almacenado por completo en memoria, pues este es procesado como una sucesión de datos, invocando al mismo tiempo los handlers definidos por el desarrollador. SAX es más simple y fácil de usar que DOM, aunque también presenta dos desventajas. Una, cuando el documento ha sido leído no existe una representación interna en memoria del mismo, por lo que si se requiere realizar algún procesamiento adicional requiere que el documento sea procesado nuevamente y dos, SAX no permite la modificación de un documento XML, limitando únicamente a su lectura.

3.9. RDF - Resource Description Framework

RDF (Resource Description Framework) es un mecanismo basado en la tecnología XML para expresar características semánticas de los datos, RDF no es un lenguaje sino un modelo estándar de la W3C para describir metadatos y ontologías[17]. Es utilizado para representar datos sobre recursos que pueden existir en la Web. A los tipos de datos de estos recursos se los denomina metadatos. Este modelo está basado en recursos, propiedades y sentencias. Una propiedad es una característica o atributo que describe a un recurso. En cambio, una sentencia representa la relación

Objeto	Atributo	Valor
http://www.w3c.org/	creado_por	#recurso_anonimo
#recurso_anonimo	nombre	Juan
#recurso_anonimo	telefono	123-456

Cuadro 3.1: *Descripción RDF de una página web*

existente entre un recurso y una propiedad, más un valor particular de dicha propiedad. Dicho valor puede ser un valor literal o a su vez un nuevo recurso. Las descripciones RDF se denominan triples o tripletes, las cuales están conformadas por un objeto (un recurso), un atributo (una propiedad), y un valor para dicho atributo. En la tabla 3.1 se muestra los tripletes necesarias para especificar una página web creada por alguien cuyo nombre es Juan y cuyo número telefónico es 123-456.

Existe la posibilidad de representar gráficamente un modelo RDF, esto se lo realiza mediante un grafo dirigido. Cada recurso se representa mediante un óvalo, y cada propiedad (arco) se representa mediante una flecha, graficando los valores literales como los rótulos de los arcos. En la figura 3.9 se observa un grafo que corresponde a los tripletes del cuadro 3.1.

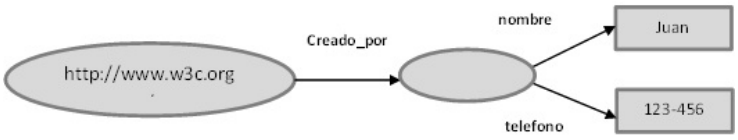

Figura 3.9: *Representación mediante un Grafo de las triplas*

Capítulo 4

Objetivos

En este apartado se expone los objetivos del presente Trabajo de Fin de Máster, en el que constan tanto el objetivo general como los objetivos específicos alineados al proyecto científico tecnológico COMPUTAPLEX, de tal manera que se dote con nuevas funcionalidades que permitan ir cubriendo las necesidades y así lograr la automatización del proceso de Experimentación en Ingeniería de Software.

4.1. Sub-Proyecto SARIDIF

Como ya se ha comentado en secciones anteriores SARIDIF - Software de Agentes para la Recuperación de Información de Diversas Fuentes, se centra en aportar a la evolución del proyecto COMPUTAPLEX mediante la implementación de componentes software que permitan extraer datos experimentales de repositorios distribuidos y disponibles en bases de datos MySQL, Hojas de Cálculos, y Ficheros Estadísticos SPSS. En cuanto a la comunicación entre los agentes clientes y agentes publicadores de la plataforma COMPUTAPLEX se planteo la necesidad de establecer un vocabulario formal (ontología) que permita la interacción entre agentes de tal forma que estos puedan entender las acciones solicitadas.

4.2. Objetivo General

A continuación se presenta el objetivo general que dirige a este trabajo de fin de máster:
El objetivo general de esta tesis de máster es aportar al desarrollo del proyecto COMPUTAPLEX con la construcción de componentes software que permitan dotar al sistema de funcionalidades para la recuperación de datos disponibles en repositorios distribuidos referentes al proceso experimental del desarrollo de Software.

Para lograr este objetivo se ha debido descomponer en objetivos específicos los cuales se presentan en la siguiente sección.

4.3. Objetivos Específicos

- Realizar el estudio tecnológico-arquitectónico de la plataforma COMPUTAPLEX, con el fin de comprender las tecnologías con las cuales se encuentra construido el software y los lineamientos que deben seguirse cumpliendo.

- Implementar componentes que permitan realizar la recuperación de datos experimentales disponibles en varias fuentes de datos para COMPUTAPLEX manteniendo los lineamientos de los modelos conceptuales creados para la unificación de repositorios experimentales independientes.

- Proporcionar a los agentes de COMPUTAPLEX la capacidad de interacción mediante el uso de una ontología que defina las acciones para la comunicación entre agentes.

- Emplear una metodología para el desarrollo del proyecto de tal manera que las funcionalidades requeridas vayan incorporándose a medida que se tiene mayor conocimiento del proceso experimental en ingeniería de software.

Capítulo 5

Metodología

En este capítulo se explica la metodología empleada para la implementación de los componentes del proyecto COMPUTAPLEX. Se describen las etapas por las que atravesó el software desarrollado. Seguidamente se da a conocer cuál fue el plan de trabajo.

5.1. Descripción de la Metodología de Desarrollo

La existencia de una amplia variedad de propuestas metodológicas para el desarrollo de software, pone en evidencia que es una actividad altamente compleja. Dicha complejidad ha causado que estos marcos de trabajo usados para estructurar, planificar y controlar el proceso de desarrollo de software vayan adoptando nuevos enfoques.

Inicialmente las metodologías de desarrollo de software denominadas tradicionales se centran en el control del proceso, estableciendo rigurosamente las actividades involucradas, los artefactos que se deben producir, y las herramientas y notaciones que se usarán[6], siguen el paradigma genérico de ingeniería de requerimientos, diseño, construcción y mantenimiento[22].

Por otro lado, recientemente ha surgido de una nueva filosofía como son las metodologías ágiles, que utilizan iteraciones del ciclo de desarrollo de software muy cortas, dando mayor valor al personal, a la colaboración con el cliente y al desarrollo incremental del software. Se destacan por mostrar su efectividad en proyectos en los cuales los requisitos son dinámicos y los tiempos de

entrega de un producto son cortos sin que ello afecte a la calidad del software[22].

En cuanto al desarrollo de los componentes de COMPUTAPLEX se determinó que por la naturaleza del proyecto, la mejor alternativa metodológica para este tipo de producto es una metodología ágil en la que sea posible la generación y entrega de prototipos que a medida que van siendo desarrollados, en cada iteración puedan ser mejorados y tras varias aproximaciones se cubran los requerimientos y necesidades de la plataforma.

Tomando en consideración estos aspectos se decidió seguir la metodología ASD[12] (Adaptive Software Development) la cual se centra en la adaptación continua a circunstancias cambiantes del proceso de trabajo en vez de la optimización, incluso se reconoce que en cada iteración se producirán cambios y hasta errores. Al igual que otras metodologías ágiles, su funcionamiento es cíclico y se compone de tres fases: Especular-Colaborar-Aprender.

En la metodología ASD las principales características son:

- Disponer de un ciclo de vida iterativo.
- Estar orientado a componentes software (funcionalidades que el producto debe poseer) y no en tareas que se va alcanzar con este objetivo.
- Es limitado en el tiempo.
- Orientado por riesgos.
- Tolerante al cambio.

En la figura 5.1 se pude apreciar el ciclo de vida que sigue esta técnica. En este caso el proyecto comienza con una fase denominada “Especular” en la que existe una planificación tentativa del proyecto en función de las entregas que se irán realizando. La siguiente fase es “Colaborar” es aquella en la que se construye la funcionalidad definida durante la especulación. La fase final es

Figura 5.1: *Metodología ASD, tomada de [HighSmith, 2000]*

"Aprender" consiste en la revisión de calidad que se realiza al final de cada ciclo.

ASD al estar basado en aproximaciones sucesivas; permitió en cada una de éstas entregar un grupo de funcionalidades o componentes, los cuales fueron construidos siguiendo el enfoque de dicha metodología la cual especifica que durante un ciclo iterativo el software debe construirse pieza por pieza[26]. Por otro lado, ASD es considerada una metodología ágil por cumplir con los principios del manifiesto ágil: 1) Individuos e interacciones sobre procesos y herramientas. 2) Software funcionando sobre documentación extensiva. 3) Colaboración con el cliente sobre negociación contractual. 4) Respuesta ante el cambio sobre seguir un plan. La ASD no prescribe tareas al momento de llevar a cabo la construcción del software, simplemente menciona que todas las prácticas que sirvan para reforzar la colaboración serán preferidas y deben tomarse como buenas prácticas de desarrollo.

Actividades del Adaptative Software Developed

En la figura 5.2 se puede observar las fases y actividades que sigue la metodología Adaptative Software Developed (ASD). A continuación se hace una descripción en detalle de cada una de estas:

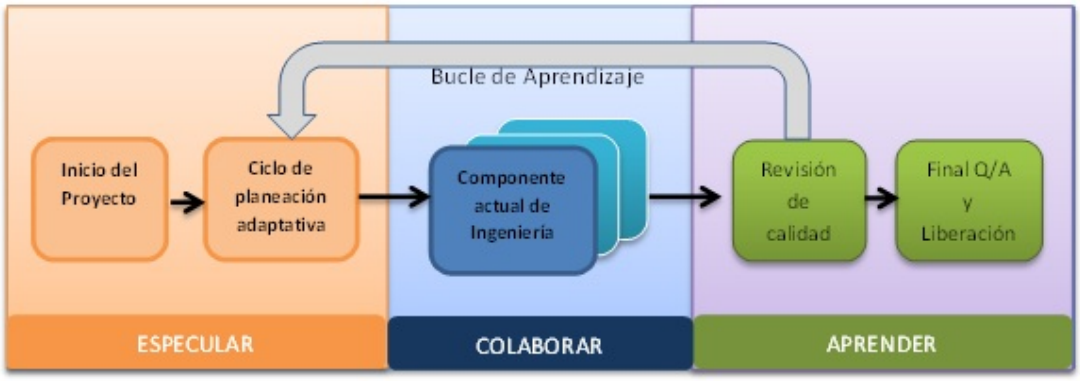

Figura 5.2: *Actividades del Software Adaptative Development (ASD)*

La primera fase de la metodología se denomina “Especular”, entre los principales aspectos se debe definir la misión, objetivos y metas del proyecto, además de tomar en consideración sus limitaciones o riesgos. Aunque una planificación temporal es considerada por la metodología, las estimaciones realizadas pueden sufrir desviaciones. Sin embargo, es importante priorizar las tareas de los trabajadores para cumplir los acuerdos dentro de los plazos. Seguidamente, se determina el número de iteraciones y la duración que tendrá cada una de estas, para luego definir los objetivos y funcionalidades de cada iteración.

En la segunda fase “Colaborar”, se revisan durante varias veces los requerimientos del proyecto hasta ser comprendidos. Consiste en la gestión y desarrollo concurrente del producto, por lo que es importante definir la forma de trabajo de acuerdo a las habilidades de los integrantes del grupo. Debiendo además de manera individual realizar críticas constructivas del trabajo de los demás en forma anónima, tomar estas críticas para mejorar y no generar resentimientos, tener la habilidad para trabajar en grupo y de la mano de los demás integrantes, y comunicar los problemas o preocupaciones que se tiene.

Finalmente en la fase de “Aprender” considera el aprendizaje continuo en cada ciclo, el cual es un elemento crítico para la eficacia de los equipos. En cada iteración se revisa la calidad del producto desde el punto de vista del cliente y del desarrollador, se realiza una gestión del rendi-

miento para evaluar lo que se ha aprendido en el empleo de los procesos utilizados y se ejecuta una evaluación de la situación del proyecto la cual es una etapa previa a la planificación de la siguiente iteración del proyecto.

5.2. Diagrama de flujo de las tareas realizadas

La figura 5.3 muestra la secuencia de actividades desarrolladas para alcanzar el producto final, esto se lo ha realizado con el fin de dividir el trabajo global en actividades más específicas de tal manera que sean más sencillas de llevarlas a cabo. La primera actividad corresponde al estudio de las tecnologías del proyecto COMPUTAPLEX, seguidamente se ha procedido a comprender el proceso experimental de Ingeniería de Software, a continuación se implementaron componentes para la recuperación de datos experimentales, posteriormente como cuarta actividad se define una ontología y finalmente se implementa la ontología de comunicación de agentes.

5.3. Plan temporal

El plan temporal para el presente proyecto se ha definido en tres ciclos tomando en consideración los lineamientos propuestos por la metodología ASD y principalmente por los requerimientos solicitados por el Grupo de Investigación de Ingeniería de Software Empírica de la Universidad Politécnica de Madrid.

Para el primer ciclo se estudia la arquitectura de COMPUTAPLEX, se realiza el proceso de instalación y configuración de los componentes tecnológicos bajo los cuales se encuentra implementado el software y son primordiales para poner en funcionamiento dicha plataforma. El objetivo del segundo ciclo corresponde a la implementación de componentes para la recuperación de fuentes de datos de procesos experimentales. Para finalmente en el tercer ciclo Definir e implementar una Ontología para la comunicación entre el agente cliente y el agente publicador de la plataforma.

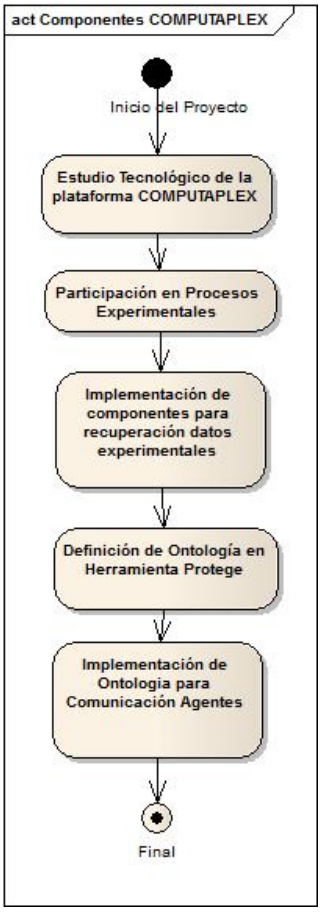

Figura 5.3: *Diagrama de flujo de las tareas proyecto*

La planificación se la puede observar en la siguiente figura 5.4, en la cual se especifica las tareas de cada uno de los tres ciclos, junto con su duración estimada, estableciendo la fecha de inicio y la fecha de finalización.

1		**Proyecto TFG**	87 days?	**11/03/14 8:00**	**9/07/14 17:00**
2		**CICLO I**	17 days?	**11/03/14 8:00**	**2/04/14 17:00**
3		Instalación de los componentes de tecnología	1 day?	11/03/14 8:00	11/03/14 17:00
4		Configuración de los componentes de tecnologia	1 day	12/03/14 8:00	12/03/14 17:00
5		Estudio de la tecnología de agentes	1 day?	13/03/14 8:00	13/03/14 17:00
6		Análisis de la arquitectura COMPUTAPLEX	7 days?	14/03/14 8:00	24/03/14 17:00
7		Pruebas	2 days?	25/03/14 8:00	26/03/14 17:00
8		Documentación del ciclo	5 days	27/03/14 8:00	2/04/14 17:00
9		**CICLO II**	27 days?	**3/04/14 8:00**	**9/05/14 17:00**
10		Estudio de lenguaje XML	2 days	3/04/14 8:00	4/04/14 17:00
11		Comprensión del proceso ISE	5 days	5/04/14 8:00	11/04/14 17:00
12		Planteamiento de la estructura XML para recuperación datos	1 day?	12/04/14 8:00	14/04/14 17:00
13		Codificación de componentes para lectura del fichero XML creado	3 days	15/04/14 8:00	17/04/14 17:00
14		Codificación de componentes para lectura de hojas de cálculo	3 days?	18/04/14 8:00	22/04/14 17:00
15		Codificación de componentes para lectura de ficheros SPSS	3 days?	23/04/14 8:00	25/04/14 17:00
16		Implementación de métodos para generar mensajes RDF/XML desde el agente publiser	3 days?	26/04/14 8:00	30/04/14 17:00
17		Pruebas	2 days?	1/05/14 8:00	2/05/14 17:00
18		Documentación del ciclo	5 days	3/05/14 8:00	9/05/14 17:00
19		**CICLO III**	13 days?	**10/05/14 8:00**	**28/05/14 17:00**
20		Realización de cambios y pruebas de componentes	2 days?	10/05/14 8:00	13/05/14 17:00
21		Instalación de Herramienta Protege	1 day?	12/05/14 8:00	12/05/14 17:00
22		Estudio de Ontologías	5 days?	13/05/14 8:00	19/05/14 17:00
23		Plantenamiento de Ontologia de acciones	3 days?	20/05/14 8:00	22/05/14 17:00
24		Implementación de Ontologia de acciones en java	2 days?	23/05/14 8:00	26/05/14 17:00
25		Pruebas	2 days?	27/05/14 8:00	28/05/14 17:00
26		Documentación del Proyecto	30 days?	29/05/14 8:00	9/07/14 17:00

Figura 5.4: *Plan temporal del proyecto*

Capítulo 6

Resultados

En las siguientes secciones se irá explicando las etapas y cada una de las actividades por las cuales fue atravesando el proyecto SARIDIF. Como se pudo observar en el capítulo anterior para el presente proyecto se ha determinado una planificación que consta de tres ciclos tomando en consideración los objetivos planteados y las actividades descritas en el diagrama de flujo de actividades. A continuación se presenta una descripción de cada uno de estos tres ciclos:

6.1. Primer ciclo

6.1.1. Especular:

En este primer ciclo se pone en marcha al proyecto, mediante la tarea de iniciación del proyecto como lo establece la metodología estudiada en el capítulo anterior, para lo cual es importante identificar la misión, la misma que se ha establecido como:
"Mediante la construcción de componentes software para la recuperación de datos experimentales de Ingeniería de Software registrados en diversas fuentes, el proyecto COMPUTAPLEX dispondrá de mejores prestaciones en cuanto a la unificación de repositorios independientes tanto a nivel conceptual como a nivel ontológico".
Seguidamente, se identifican dos roles para el equipo del proyecto: rol desarrollador y rol cliente. El rol desarrollador lo cumplirá el tesista y el rol cliente los miembros del grupo GRISE.

El objetivo principal de este primer ciclo corresponde a comprender la plataforma COMPUTAPLEX, siendo necesario estudiar la arquitectura sobre la que se encuentra implementada; así como también, realizar la instalación los componentes tecnológicos necesarios para poner en funcionamiento la plataforma. Además, fue necesario estudiar el paradigma orientado a agentes, con el fin de comprender los fundamentos teóricos para luego poder implementarlos en el lenguaje java, por medio del uso del framework JADE, el cual es el entorno tanto de ejecución como de implementación de dicha Tecnología.

COMPUTAPLEX es un proyecto de investigación científica tecnológica aún en construcción, al cual el presente proyecto aporta con la implementación de componentes para la recuperación de datos experimentales. La ejecución del proyecto, en esta fase de la metodología ASD partió con unos requerimientos que fueron comprendiéndose en la medida que el proceso experimental iba entendiéndose, así como también las tecnologías con las cuales debía ser desarrollado cada uno de estos componentes. Todo ello permitió ir determinado cuales eran las prioridades a ser implementadas, la estimación del número de iteraciones a necesitarse y el tiempo de duración de cada una de ellas.

6.1.2. Colaborar:

Como se ha mencionado anteriormente en el plan de trabajo, la primera actividad corresponde al estudio tecnológico de la plataforma COMPUTAPLEX, requiriendo para este fin llevar a cabo ciertas tareas entre los participantes del proyecto en relación a la instalación y configuración de los componentes de tecnología en el ordenador que será utilizado por el desarrollador. Específicamente estas tareas fueron: Instalación de la herramienta informática de desarrollo Eclipse Kepler. Descarga del código fuente del proyecto desde el sistema de control de versiones SVN. Configuración de librerías empleadas por la plataforma COMPUTAPLEX necesarias para la puesta en

funcionamiento.

Una vez que se tuvo acceso a la aplicación se estudió la estructura arquitectónica de la plataforma, así como también, de sus tecnologías empleadas, entre las cuales se destacan:

- Tecnología de sistemas de agentes software.
- Estudio de patrones de diseño.
- Estudio de los marcos de trabajo JADE y JENA.
- Estudio del API DocumentBuilder para el trabajo con ficheros XML.
- Estudio del API para la lectura de Hojas de cálculo y archivos SPSS.

Otra de las actividades importantes en este primer ciclo fue la participación real del proceso experimental durante la asignatura de Ingeniería de Software Experimental, en la que se constató la manera de llevar a la práctica los principios teóricos de una investigación experimental de Ingeniería de Software, con el fin de llegar a obtener un conocimiento global de lo que este proceso conlleva. Y de esta manera identificar las condiciones de aplicabilidad, los procedimientos, los materiales e instrumentos necesarios para desarrollar cada una de las etapas por las que atraviesa el proceso y la información generada en cada una de ellas.

Por otro lado, en esta primera etapa se ha considerado realizar el diagrama de agente de la plataforma COMPUTAPLEX, en el cual se toma en consideración los roles, servicios ofertados, planes (intensiones), objetivos/acciones (deseos) y conocimiento (creencias), esta información se puede apreciar en los cuadros 6.1 y 6.2.

En cuanto a la identificación de las funciones del agente o los servicios que este oferta se utilizó el diagrama de servicios de agentes (ver los cuadros 6.3 y 6.4) en el cual es necesario identificar para cada servicio: las entradas, salidas, pre-condiciones y post-condiciones.

Agente Cliente
Roles
-Agente cliente
-Agente publicador
Servicios
-Solicitar información experimental:Agente publicador
-Receptar información experimental
-Procesar mensajes RDF/XML
-Unificar información experimental
Creencias
-Información de la ontología de acciones del acto comunicativo.
-Información de agentes publicadores registrados.
Deseos
-Localizar agentes publicadores.
-Generar solicitudes a agentes publicadores.
Intensiones
-Administrar información experimental.
-Permitir toma de decisiones.

Cuadro 6.1: *Diagrama de roles de agente cliente*

Agente Publicador
Roles
-Agente publicador
-Agente cliente
Servicios
-Atender solicitud de datos experimentales
-Recuperar datos fuente de datos
-Interpretar descripciones
-Generar ontologías dinámicas RDF/XML
Creencias
-Información de fuentes de datos.
-Información de ontología de acciones para el acto comunicativo.
Deseos
-Recuperar datos de las fuentes asociadas al agente.
-Transmitir respuestas mediante mensajes RDF/XML.
Intensiones
-Gestionar fuente de datos.
-Generar modelos ontológicos.

Cuadro 6.2: *Diagrama de roles de agente publicador*

6.1.3. Aprender:

Como resultado de la fase de aprendizaje se ha conseguido asimilar los nuevos conocimientos sobre el paradigma orientado agentes y las tecnologías con las que se encuentra implementado el proyecto COMPUTAPLEX. De tal manera que los posteriores ciclos planificados para el desarrollo de los componentes del presente proyecto se puedan cubrir de manera satisfactoria.

Diagrama de servicios de agente cliente				
Servicio	**Entrada**	**Salida**	**Pre-condición**	**Post-condición**
Generar solicitud del listado de agentes	Directory Facilitator (DF)	Lista de agentes publicadores	Plataforma ejecutándose - agentes registrados	Se dispone del listado de agentes publicadores
Emitir solicitud a cada agente publicador	lista de agentes publicadores	Colección de mensajes RDF/XML con información experimental	comunicación activa entre agentes	Conjunto de mensajes RDF/XML para unificar los datos experimentales

Cuadro 6.3: *Diagrama de servicios de agente cliente*

Diagrama de servicios de agente publicador				
Servicio	**Entrada**	**Salida**	**Pre-condición**	**Post-condición**
Recuperación datos experimentales de fuentes de datos	Parámetros de configuración de fuentes, parámetros del experimento	registros de la consulta de información experimental	Fuente de datos exista/activa/disponible	colección de datos experimentales recuperados de la fuente
Generación de mensaje de respuesta a solicitud experimental	Colección de datos experimentales recuperados de la fuente de datos	String que encapsula el modelo de representación de datos experimentales mediante ontología	registros de datos experimentales	generación de mensajes RDF/XML en contestación de solicitud agente cliente.

Cuadro 6.4: *Diagrama de servicios de agente publicador*

En cuanto a la fase final de este primer ciclo se ha constatado que la configuración realizada para la ejecución tanto de la plataforma como del agente cliente y del agente publicador es la adecuada, puesto que se pudo verificar que existió comunicación entre agentes por medio de la transmisión de mensajes RDF/XML a partir de los datos extraídos desde la fuente de datos MySQL, como inicialmente constaba en el proyecto, superando así la evaluación de prueba.

6.2. Segundo ciclo

6.2.1. Especular:

Para el segundo ciclo fue necesario repasar el lenguaje XML (Lenguaje de Marcado Extensible) con el objetivo de plantear una estructura propia basada en tags XML, que permita especificar

tanto la localización como los datos experimentales disponibles dentro de hojas de cálculo y ficheros SPSS que contienen resultados de experimentadores del área de ingeniería de software. En este mismo ciclo, se debe conocer el uso de APIs para la lectura de datos de las hojas de cálculo y ficheros SPSS.

Los requerimientos funcionales del segundo ciclo se presentan en el cuadro 6.5. En ella se lista el requerimiento y su correspondiente descripción.

Requerimiento	Descripción del Requerimiento
RQ 01	La plataforma debe tener un agente cliente que envía peticiones a los agentes publicadores que se encuentran registrados en el DF de la plataforma COMPUTAPLEX.
RQ 02	El agente cliente estará en estado de espera hasta que los agentes Publicadores envíen la respuesta a sus solicitudes generadas.
RQ 03	La plataforma COMPUTAPLEX debe ser capaz de poseer un mecanismo para la recuperación de datos experimentales.
RQ 04	El acceso a los datos experimentales se lo realizará implementado métodos de una interfaz disponible en la arquitectura COMPUTAPLEX, los métodos corresponden con la obtención de los experimentos, factores, niveles entre otros.
RQ 05	La plataforma debe poseer la capacidad para extender nuevas clases que permitan extraer datos de nuevas fuentes sin alterar los actuales mecanismos establecidos.
RQ 06	Se debe definir una estructura de información en XML que contiene elementos obligatorios y opcionales para describir los datos experimentales disponibles en hojas de cálculo.
RQ 07	Para la Inicialización del RMA (Remote Agent Management), de los agentes y de la plataforma se pasará como argumentos del programa los nombres de los archivos XML de configuración.
RQ 08	La aplicación realizará un mapeo desde los archivos de configuración XML hacia una lista de objetos para la posterior recuperación de la información experimental (e.g nombre de experimentos, factores, niveles, etc.) desde la fuente de datos.
RQ 09	La aplicación debe ser capaz de extraer la información disponible en hojas de cálculo de acuerdo al listado de objetos que contienen la información paramétrica de los ficheros y su contenido referente a datos experimentales.
RQ 10	La aplicación debe ser capaz de extraer la información experimental contenida en ficheros de análisis estadístico SPSS.

Cuadro 6.5: *Requisitos Funcionales del segundo ciclo*

Los requerimientos no funcionales del segundo ciclo se presentan en el cuadro 6.6. En ella se lista el requerimiento y su correspondiente descripción.

Requerimiento	Descripción del Requerimiento
RNF 01	El lenguaje en el que deben ser desarrollados los componentes es JAVA.
RNF 02	La codificación de los métodos deben seguir la misma codificación presente en el proyecto COMPUTAPLEX.
RNF 03	El API con la que se realizará el parseo de archivos XML a clases Java puede ser SAX o DOM.
RNF 04	Para la recuperación de información desde hojas de cálculo se debe utilizar un API que provea estas funcionalidades.
RNF 05	El entorno de desarrollo para este proyecto será ECLIPSE.
RNF 06	Para la programación del proyecto se debe hacer uso de un sistema de control de versiones SVN.

Cuadro 6.6: *Requisitos No Funcionales del segundo ciclo*

6.2.2. Colaborar:

En cuanto a esta fase se procedió con la construcción de los componentes de ingeniería, es decir se fue realizando la codificación en el lenguaje de programación Java, el cual es el lenguaje en el que está desarrollado la Plataforma COMPUTAPLEX. El resultado de esta fase es un prototipo funcional que tras ser evaluado, en la fase siguiente de la metodología puede pasar las pruebas o bien crear una nueva iteración donde se corrija las funcionalidades no superadas.

Para la implementación de componentes para la recuperación de datos experimentales, fue necesario plantear una estructura XML capaz de describir la información correspondiente a los resultados experimentales que se encuentran registrados en fuentes de datos como son hojas de cálculo. Para este fin se debió determinar la forma en que la información está distribuida en las hojas de cálculo, para así mediante un fichero de configuración XML, determinar la localización y opcionalmente el nombre del elemento que representa la información referente a un experimento, sus factores o sus niveles. De igual manera algo similar fue realizado para los ficheros SPSS.

En la figura 6.1 se muestra la arquitectura obtenida en este segundo ciclo, en la cual se puede observar gráficamente que el mecanismo de recuperación de datos de las fuentes de datos no únicamente da acceso a las bases de datos MySQL sino que también se incorporan Hojas de

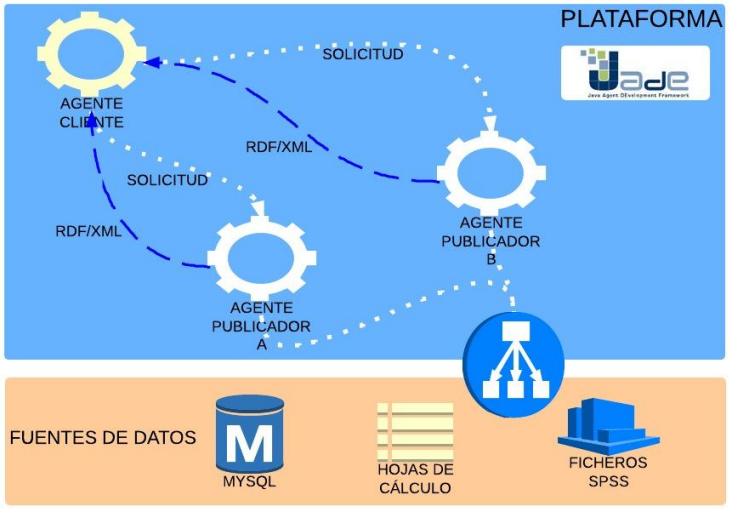

Figura 6.1: *Arquitectura de Computaplex - Segundo Ciclo*

cálculo y ficheros estadísticos SPSS. A continuación se hace una descripción de esta etapa de implementación de componentes.

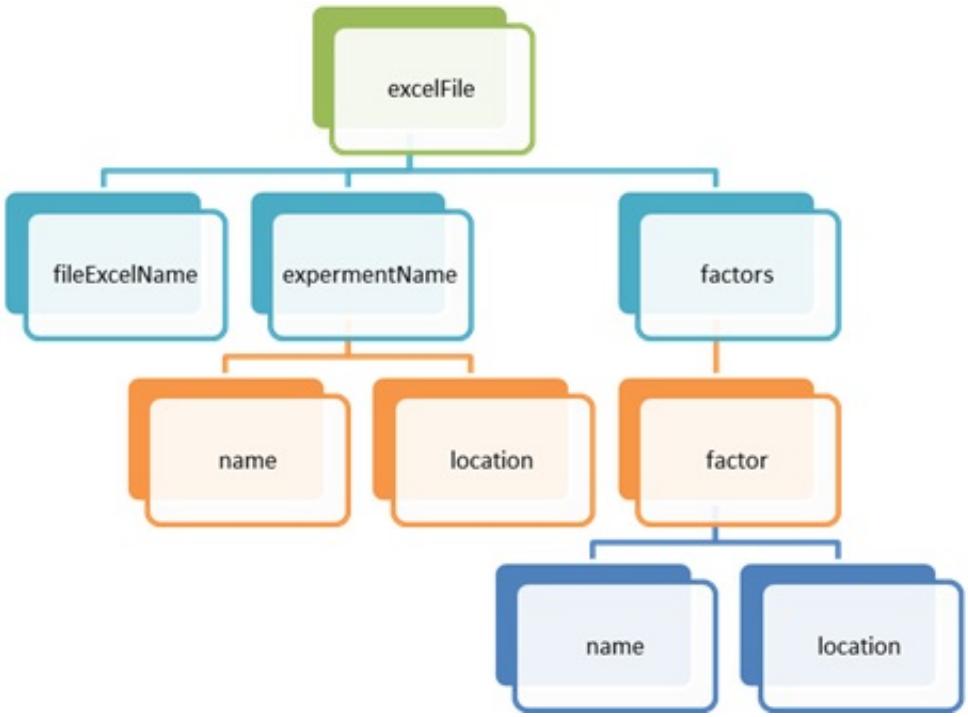

Figura 6.2: *Esquema para el archivo de configuración XML*

Antes de codificar los componentes para la recuperación de datos de la aplicación, fue necesario plantear la estructura XML del archivo de configuración, la misma que se puede apreciar en la figura 6.2, y en la que constan los tags necesarios para describir una hoja de cálculo que pueden contener uno o varios experimentos.

En la figura 6.3 se puede apreciar una instancia del archivo de configuración XML. El mismo que se describe a continuación. Inicialmente fue necesario realizar la definición de la estructura XML para describir la información experimental contenida en fuentes de datos. En la misma se puede apreciar que el nodo principal contiene varias etiquetas <*excelFile*>, cada una de estas sirve para representar a una hoja de cálculo, por lo que es necesario conocer el nombre del fichero mediante la etiqueta <*fileExcelName*>, seguidamente la etiqueta <*experimentName*> posee una estructura para describir el nombre y localización del experimento, la siguiente etiqueta <*factors*>

```
<?xml version="1.0"?>
    <computaplexExcel>
        <excelFile>
            <fileExcelName>AllData.xls</fileExcelName>
            <experimentName>
               <name>UPM-2005</name>
               <location>0</location>
            </experimentName>
            <factors>
                <factor>
                   <name>Aplicacion</name>
                   <location>4</location>
                </factor>
                <factor>
                   <location>9</location>
                </factor>
            </factors>
        </excelFile>
        <excelFile>
            <fileExcelName>Experimento_2013.xls</fileExcelName>
            <experimentName>
               <name>ORT-2013</name>
               <location>0</location>
            </experimentName>
            <factors>
                <factor>
                   <name>App</name>
                   <location>6</location>
                </factor>
            </factors>
        </excelFile>
    </computaplexExcel>
```

Figura 6.3: *Instancia de la definición de la estructura XML*

define una estructura que permite contener cuantos factores disponga dicho experimento, como se puede apreciar la etiqueta <*factor*> incluye en algunos casos dos etiquetas:<*name*> y <*location*>, esto se debe a que la etiqueta <*name*> es una etiqueta opcional y en el caso de no definir esta información será extraída de la ubicación correspondiente dentro del fichero especificado.

Posteriormente, mediante el uso de las APIs del paquete javax.xml.parsers se realiza un mapeo desde el fichero de configuración XML hacia una clase java, de tal forma que coincida cada uno de los descriptores XML con los atributos de esta clase. A partir de la información de estos objetos se realiza la lectura de cada hoja de cálculo mediante el API JExcelAPI. Estos datos extraídos son encapsulados en una Ontología para que puedan ser enviados por medio de un mensaje al agente solicitante de dicha información. Los métodos implementados para recuperar los experimentos, sus factores y sus niveles siguen la misma lógica de programación que el segmento de código del cuadro 6.7 en donde se puede observar el proceso descrito anteriormente.

```
@Override
public String getFactorsLevels(String agentName, String nameExperiment)
    {

 StringWriter sw = new StringWriter();
 OntModel m = ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);
 String expNS = "http://main.grise.upm.es/empiricalStudies#";
 String coorNS = "http://main.grise.upm.es/coordination#";
 String agentNS = "agent://" + agentName + "#";
 try {
  for(int i=0;i<filesConfiguration.size();i++){
   ExcelDataFile exlsData=new ExcelDataFile(filesConfiguration.get(i));
   exlsData.leerArchivoExcel("levels",nameExperiment);
   for(String factores:exlsData.getParametroRetorno()){
     OntClass o = m.createClass(expNS + "Experiment");
         Individual experiment= m.createIndividual(agentNS+factores, o)
            ;
         Property name = m.createProperty( expNS + "valueLevels" );
         m.add(experiment, name, factores);
         Property agent = m.createProperty( expNS + "agent" );
         m.add(experiment, agent, agentNS);
        }
   }
 }catch (Exception e) {
   OntClass o = m.createClass(coorNS + "Exception");
   Individual error = m.createIndividual(agentNS + "error", o);
   Property origin = m.createProperty(coorNS + "origin" );
   m.add(error, origin, "database");
 }
 m.write(sw);
 return sw.toString();
}
```

Cuadro 6.7: *Segmento de código del método obtener niveles de un experimento*

En la figura 6.4 se puede apreciar la secuencia de mensajes que se da dentro de la plataforma. El agente cliente es quien inicia la petición del listado de los agentes publicadores al Directory Facilitator (DF), una vez que se devuelve dicha información, el agente cliente emite una solicitud a cada agente publicador registrado, para que extraiga la información experimental de la fuente de datos asociado a este. Es importante mencionar que la fuente de datos en el diagrama representa una base de datos MySQL, hojas de cálculo o ficheros estadísticos SPSS.

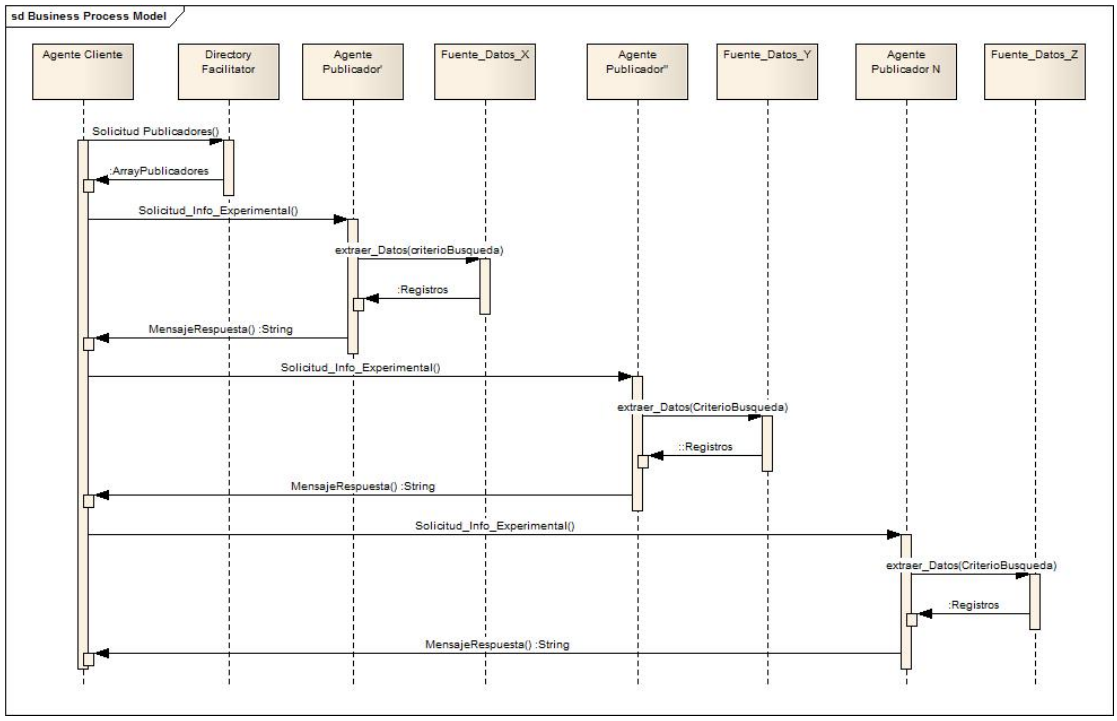

Figura 6.4: *Diagrama de secuencia de mensajes entre Agentes*

6.2.3. Aprender:

Una vez que se han logrado implementar los requerimientos funcionales correspondientes al segundo ciclo, se procede con la verificación mediante pruebas que permitan constatar que el prototipo obtenido, cumple correctamente con las funcionalidades requeridas y los componentes desarrollados se integran adecuadamente con la plataforma COMPUTAPLEX. En el cuadro 6.8

se muestra el detalle de la verificación llevada a cabo y en la cual se observa que se cumplió con éxito los requerimientos de este segundo ciclo.

Criterio	**SI/NO/NA**
¿La plataforma tiene un agente cliente que envía peticiones a los agentes publicadores que se encuentran registrados en el DF de la plataforma COMPUTAPLEX?	SI
¿El agente cliente espera hasta que los agentes Publicadores envíen la respuesta a sus solicitudes generadas?	SI
¿La plataforma COMPUTAPLEX posee un mecanismo para la recuperación de datos experimentales?	SI
¿El acceso a los datos experimentales se lo realiza implementado métodos de una interfaz disponible en la arquitectura COMPUTAPLEX, los métodos corresponden con la obtención de los experimentos, factores, niveles entre otros?	SI
¿La plataforma posee la capacidad para extender nuevas clases que permitan extraer datos de nuevas fuentes sin alterar los actuales mecanismos establecidos?	SI
¿Se ha definido una estructura de información en XML que contiene elementos obligatorios y opcionales para describir los datos experimentales disponibles en hojas de cálculo?	SI
¿La Inicialización del RMA (Remote Agent Management), de los agentes y de la plataforma se realiza pasando como argumentos del programa los nombres de los archivos XML de configuración?	SI
¿La aplicación realiza un mapeo desde los archivos de configuración XML hacia una lista de objetos para la posterior recuperación de la información experimental (e.g nombre de experimentos, factores, niveles, etc.) desde la fuente de datos?	SI
¿Para el procesado de documentos XML se hace uso de las APIs disponibles en el paquete javax.xml.parser?	SI

Cuadro 6.8: *Verificación de Requisitos Funcionales del segundo ciclo*

6.3. Tercer ciclo

6.3.1. Especular:

El tercer ciclo de construcción de los componentes software tiene como objetivo la definición de una ontología que permita formalizar el acto de comunicación entre el agente cliente y el agente publicador, para que así ambos agentes compartan el mismo significado para los conceptos que se están manejando dentro de la plataforma COMPUTAPLEX, en el contexto de la información referente a procesos experimentales dentro del área de la Ingeniería de Software.

Para este ciclo se hace necesario investigar la herramienta de diseño de ontologías Protégé, puesto que es un editor de código abierto ampliamente utilizado para el diseño de ontologías y

representación de conocimiento, además de proveer opciones para agregar componentes adicionales. De manera especial se ha seleccionado esta herramienta puesto que dispone del plug-in Ontology Bean Generator el cual permite a partir del diseño ontológico la generación automática de las clases java que deben ser agregadas al proyecto para su posterior uso.

En el cuadro 6.9 se describen cada uno de los requerimientos funcionales correspondientes al tercer ciclo de desarrollo del proyecto.

Requerimiento	Descripción del Requerimiento
RQ 01	La plataforma COMPUTAPLEX debe poseer una representación Ontológica del concepto Experimento.
RQ 02	La plataforma COMPUTAPLEX debe poseer una representación Ontológica del concepto Factor de un experimento en específico.
RQ 03	La plataforma COMPUTAPLEX debe poseer una representación Ontológica del concepto Nivel de un experimento específico.
RQ 04	La plataforma COMPUTAPLEX debe poseer una representación Ontológica de la acción ObtenerExperimento.
RQ 05	La plataforma COMPUTAPLEX debe poseer una representación Ontológica de la acción ObtenerFactor.
RQ 06	La plataforma COMPUTAPLEX debe poseer una representación Ontológica del concepto ObtenerNivel.
RQ 07	La aplicación debe realizar el acto comunicativo entre el agente cliente y el agente publicador mediante el uso de la Ontología definida.

Cuadro 6.9: *Requisitos Funcionales del tercer ciclo*

Los requerimientos no funcionales del tercer ciclo se presentan en el cuadro 6.10. En ella se lista el requerimiento y su correspondiente descripción.

6.3.2. Colaborar:

Otra actividad realizada en el presente trabajo es la definición de una Ontología, la cual permite realizar un acto de comunicación entre el agente cliente y el agente publicador. Este proceso se lo realizó por medio de la herramienta Protégé.

Requerimiento	Descripción del Requerimiento
RNF 01	La definición de la ontología debe realizarse en la herramienta Protégé.
RNF 02	La ontología creada debe estar escrita en código JAVA.
RNF 03	La ontología generada deberá incorporarse dentro de la capa de Servicios.
RNF 04	La implementación del código de la ontología debe realizarse en el entorno de desarrollo eclipse.
RNF 05	La performativa utilizada debe ser la misma que se utilizaba antes de disponer de la ontología.
RNF 06	La programación de estos componentes del proyecto se debe hacer uso de un sistema de control de versiones SVN.

Cuadro 6.10: *Requisitos No Funcionales del segundo ciclo*

Finalmente se realizó la implementación de la ontología creada en la plataforma COMPUTAPLEX, para este fin se configuró el plug-in llamado Ontology Bean Generator for Jade, el cual genera el código java partiendo de la definición ontológica creada en la etapa anterior. Posteriormente se codificó los segmentos de código que hacen uso de la ontología para realizar la comunicación entre los agentes de la plataforma.

Construcción de la Ontología utilizando Protégé

El proceso de creación de las ontologías se lo desarrolla de manera iterativa, esto es que una vez definida la ontología inicial esta evoluciona y se va refinando a través de sucesivas iteraciones para lograr un mayor nivel de detalle.

En el presente trabajo para la construcción de la ontología se utilizó la metodología propuesta por Noy en[23], la cual establece siete pasos a seguir: 1) Determinar el dominio y alcance de la ontología, 2) Considerar la reutilización de ontologías existentes, 3) Enumerar términos importantes para la ontología, 4) Definir las clases y la jerarquía de clases, 5) Definir las propiedades de las clases: slots, 6) Definir las facetas de los slots. 7) Crear instancias. Cabe mencionar que no todos estos paso fueron utilizados por cuanto nuestro interés no se centra en poseer una base de conocimiento, sino más bien únicamente la estructura jerárquica de los conceptos y relaciones que permitan llevar a cabo el acto comunicativo entre los agentes de la plataforma COMPUTAPLEX.

El ámbito o dominio de aplicación de la ontología está relacionada con en el proceso experimental de la Ingeniería de Software. En cuanto a las preguntas en las que los agentes pueden basar la comunicación son: ¿Cuáles son los nombres de los experimentos realizados?. Para un experimento en específico, ¿Qué factores se analizaron en la investigación?. Para un experimento en específico, ¿Cuáles fueron los niveles de la investigación?.

Posteriormente, se realizó una revisión de Ontologías que traten sobre temas referentes a la experimentación de ingeniería de software sin tener resultados satisfactorios, por lo que la reutilización de ontologias en el presente trabajo no fue posible.

En cuanto a la enumeración de la terminología, que permita identificar los conceptos necesarios para definir la ontologia se realizó un revisión del proceso experimental de ingeniería de software, obteniendo así la abstracción de la terminología que representa el conocimiento dentro de este campo. Entre los términos seleccionados de este análisis son: Experimento, Factor y Nivel. En la figura 6.5 se muestra la jerarquía de la ontología creada.

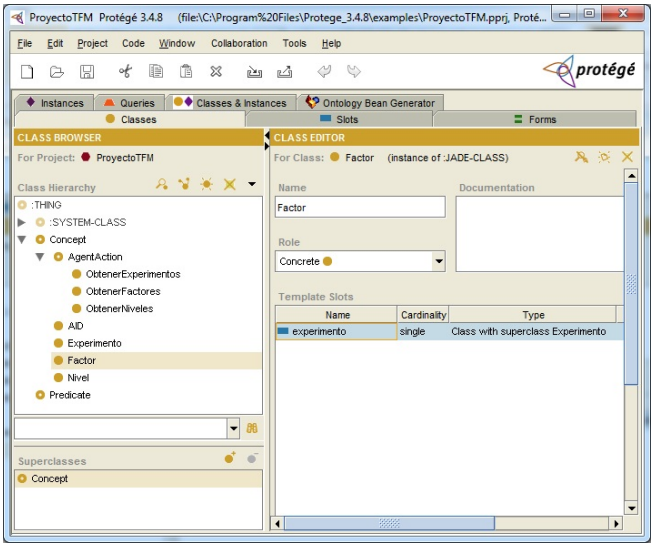

Figura 6.5: *Jerarquía de clases representada en Protégé*

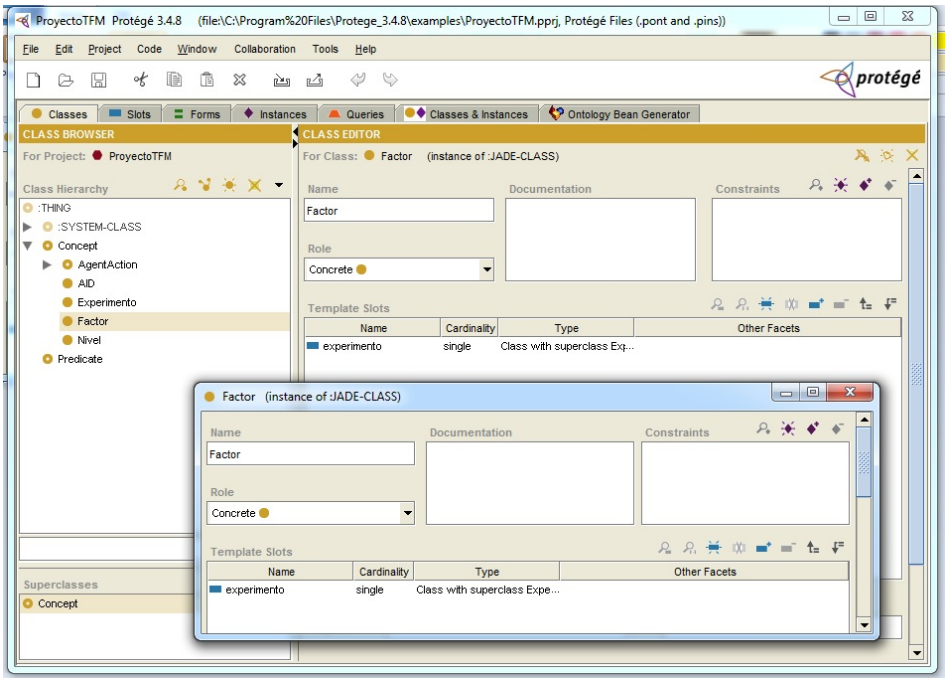

Figura 6.6: *Slots para el concepto Factor*

Además fue necesario definir las propiedades de los conceptos, a los cuales Protégé los denomina slots. Un slot una vez creado queda registrado en el proyecto, con lo que puede ser reutilizado por otras clases sin necesidad de volver a crearlas. En la figura 6.6 se puede observar el formulario donde son creados los slots, para este caso se puede apreciar la creación de un slot para el concepto Factor.

Una vez obtenida la definición de la Ontología se procedió a instalar el plug-in Ontology Bean Generator el cual se encarga de generar las clases java de la Ontología creada, con estas clases se procede a importar dentro del proyecto COMPUTAPLEX para hacer uso de dichos conceptos y acciones como acto comunicativo entre el agente cliente y el agente publicador. En la figura 6.7 se puede observar la configuración para la exportación de la Ontología mediante el plug-in de la herramienta Protégé.

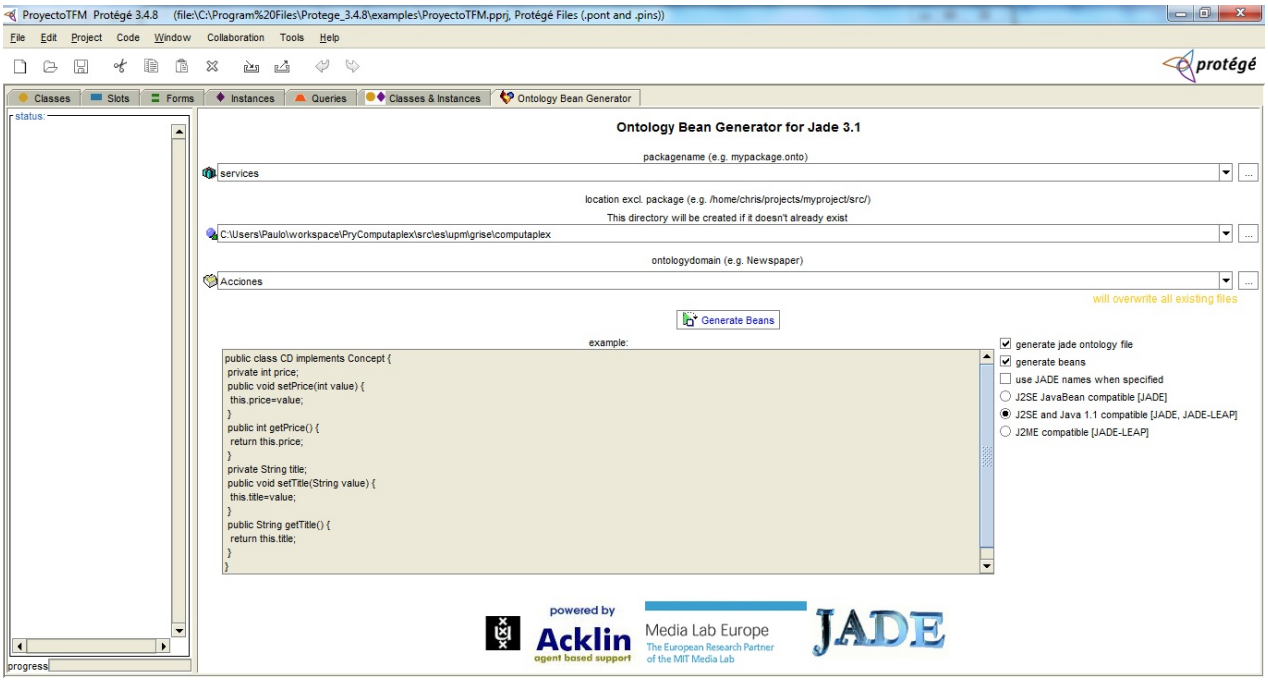

Figura 6.7: *Generación de Beans Java en Protégé*

6.3.3. Aprender:

Una vez que se han logrado definir e implementar la ontología para la comunicación entre agentes de la plataforma, se procede con la verificación mediante pruebas que permitan constatar que el acto comunicativo se desarrolla adecuadamente y que los componentes desarrollados se integran adecuadamente con la plataforma COMPUTAPLEX, este proceso se lo describe a continuación.

Verificación del proceso de comunicación entre agentes

Para la verificación se ha procedido a realizar las siguientes actividades que se describen a continuación: En el segmento de código mostrado en el cuadro 6.11 se muestra el contenido del archivo de configuración XML que describe a una hoja de cálculo junto con su información experimental. Entre las etiquetas se puede observa el nombre del fichero de la hoja de cálculo, el nombre del experimento junto con su ubicación y la localización de los factores de dicho experimento.

```
<?xml version="1.0"?>
        <computaplexExcel>
                <excelFile>
                        <fileExcelName>AllData.xls</fileExcelName>
                        <experimentName>
                           <name>UPM-2005</name>
                           <location>0</location>
                        </experimentName>
                        <factors>
                               <factor>
                               <location>9</location>
                            </factor>
                        </factors>
                </excelFile>

        </computaplexExcel>
```

Cuadro 6.11: *Estructura XML para describir información experimental contenida en hojas de cálculo*

En el acto comunicativo entre agentes de la plataforma COMPUTAPLEX, parte de una solicitud desde el agente cliente hacia los agentes publicadores registrados hasta el momento en la plataforma, para que estos a su vez recuperen la información correspondiente a dicha solicitud desde la fuente de datos que llevan asociada. El segmento de código del cuadro 6.12 muestra la solicitud de niveles de un experimento procedente de un agente cliente, en dicha solicitud se debe establecer la ontología y la codificación definida para llevar a cabo la comunicación. El mismo procedimiento se ha implementado para tratar las otros tipos de solicitudes de este proyecto, los cuales se pueden ver en la sección de Anexos.

Para realizar la recepción de la solicitud emitida por el agente cliente y atender dicho pedido, los agentes publicadores necesariamente deben compartir y establecer tanto la ontología como la codificación empleada. Esto se puede observar en el segmento de código del cuadro 6.13. En dicho código se aprecia que se realiza una comprobación del codec y la ontología para posteriormente hacer una verificación del tipo de performativa utilizada para con ello extraer el contenido del

```
          ObtenerNiveles se=new ObtenerNiveles();
          Experimento ex=new Experimento();
          ex.setNombre("ORT");
          se.setExperimento(ex);
          se.setAgente("agente 2");
          msg.setLanguage(codec.getName());
          msg.setOntology(ontologia.getName());
          try{
            getContentManager().fillContent(msg, se);
          } catch (CodecException | OntologyException e) {
            // TODO Auto-generated catch block
                  e.printStackTrace();
          }
          myAgent.send(msg);
```

Cuadro 6.12: *Segmento de código para la solicitud de niveles del agente cliente*

mensaje el cual corresponde a una instancia de la clase de solicitud al cual debe atender el agente, para así iniciar con la tarea de extracción de información de la fuente de datos.

Como respuesta a la solicitud, el agente cliente recibe un mensaje RDF/XML el cual contiene la información extraída de la fuente de datos asociada al agente publicador. El segmento de código del cuadro 6.14 muestra la respuesta de los niveles obtenidos del experimento especificado en el archivo de configuración XML.

En la sección de Anexos se encuentra el código fuente completo de todo el proceso de comunicación, en el que constan los diferentes tipos de solicitudes que pueden atender los agentes publicadores.

```
public void action() {
   MessageTemplate mt = MessageTemplate.and(
   MessageTemplate.MatchLanguage(codec.getName()),
   MessageTemplate.MatchOntology(ontologia.getName()));
   ACLMessage  msg = blockingReceive(mt);

   if(msg!=null){
        if(msg.getPerformative()==ACLMessage.NOT_UNDERSTOOD){
          System.out.println("Mensaje NOT UNDERSTOOD recibido");
        }
        else{
          if(msg.getPerformative()==ACLMessage.REQUEST){
                try {
                        ContentElement ce=getContentManager().
                           extractContent(msg);
            if(ce instanceof ObtenerNiveles){
                     ObtenerNiveles sn=(ObtenerNiveles)ce;
                         ACLMessage reply = msg.createReply();
                         reply.setPerformative(ACLMessage.INFORM);
                         String content = dataAccess.getFactorsLevels(
                            myAgent.getAID().getName(),
             sn.getExperimento().getNombre());
                         reply.setContent(content);
                         myAgent.send(reply);
                        }

                } catch (UngroundedException e) {  e.printStackTrace();
                } catch (CodecException e) {  e.printStackTrace();
                } catch (OntologyException e) {   e.printStackTrace();
                }
   }
}
```

Cuadro 6.13: *Segmento de código del agente publicador para atender solicitudes*

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:j.0="http://main.grise.upm.es/empiricalStudies#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" >
  <rdf:Description rdf:about="http://main.grise.upm.es/empiricalStudies
     #Experiment">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Class"/>
  </rdf:Description>
  <rdf:Description rdf:about="agent://Paulo-PC-Publisher@GRISE#
     functional">
    <j.0:agent>agent://Paulo-PC-Publisher@GRISE#</j.0:agent>
    <j.0:valueLevels>functional</j.0:valueLevels>
    <rdf:type rdf:resource="http://main.grise.upm.es/empiricalStudies#
       Experiment"/>
  </rdf:Description>
  <rdf:Description rdf:about="agent://Paulo-PC-Publisher@GRISE#
     structural">
    <j.0:agent>agent://Paulo-PC-Publisher@GRISE#</j.0:agent>
    <j.0:valueLevels>structural</j.0:valueLevels>
    <rdf:type rdf:resource="http://main.grise.upm.es/empiricalStudies#
       Experiment"/>
  </rdf:Description>
</rdf:RDF>
```

Cuadro 6.14: *Respuesta RDF/XML receptada por el agente cliente*

Capítulo 7

Discusión y Conclusiones

7.1. Discusión

Actualmente existen varios centros de investigación e innovación que se encuentran trabajando en los diferentes métodos de investigación propios de la ingeniería de software experimental como son: encuesta, estudio de caso, cuasi-experimento, experimento controlado, con el objetivo de tener evidencias que sustenten la práctica profesional en Ingeniería de Software. Dichos experimentos, para tener validez, requieren de un número mínimo de sujetos experimentales por lo que la distribución de resultados y el trabajo cooperativo se ha convertido en una estrategia para solventar esta dificultad.

Es importante puntualizar que después del proceso experimental la información generada debe ser procesada/almacenada en diferentes medios sean estos bases de datos, hojas de cálculo o ficheros de análisis de datos. Estos mecanismos de representación de datos, varían en su estructura interna de acuerdo a los responsables del proceso experimental, presentándose de este modo dificultades en cuanto a la heterogeneidad de formatos, lo que representa una de las mayores dificultades a ser resueltas a la hora de compartir y procesar dichos datos de los repositorios independientes, por lo que una alternativa de solución viable fue el planteamiento de una estructura XML para describir y representar los datos que luego mediante ontologías y sistemas multiagentes serán compartidos entre los grupos de investigación.

En relación con el objetivo de realizar el estudio tecnológico-arquitectónico de la plataforma COMPUTAPLEX, se pudo adquirir conocimientos sobre la tecnología de agentes y el framework de implementación JADE, la importancia del uso de los patrones de diseño para crear componentes software más flexibles y reutilizables, el potencial del lenguaje XML para representar información y recursos, desarrollar software más parametrizable y también para lograr interoperabilidad entre nodos distribuidos.

En cuanto al objetivo de implementar componentes para la recuperación de datos experimentales disponibles en Hojas de cálculo y ficheros SPSS, se pudo cubrir mediante la definición de una estructura basada en tags XML que describen dichos datos, facilitando de este modo la parametrización y configuración de fuentes externas asociadas a cada agente publicador, de tal forma que la información extraída siga manteniendo los lineamientos de los modelos conceptuales (Ontologías) creados en COMPUTAPLEX y a través de estos se logre la unificación de repositorios independientes pertenecientes a los diferentes grupos de investigación.

Con respecto al objetivo de definir la Ontología de acciones para el acto comunicativo entre agentes del Proyecto COMPUTAPLEX, se pudo alcanzar sustituyendo el intercambio de mensajes que era realizado mediante cadenas de texto String, en los cuales es difícil obtener directamente la información contenida dentro de estos, por una Ontología de acciones la cual tanto el agente emisor como el agente receptor la comparten, además de conocer la manera como la información a ser transmitida mediante un mensaje ACL se encuentra encapsulada y la forma de extraer estos datos que se encuentran estructurados bajo un esquema que agrega semántica y puede ser interpretado por los agentes.

En relación con la metodología de desarrollo empleada, ASD fue seleccionada para este proyecto puesto que ofrece bondades en cuanto a la sencillez y flexibilidad en su aplicabilidad, esto

se debe a su naturaleza ágil centrado en el desarrollo iterativo e incremental. Con ello se logró que, para cada uno de los ciclos se afinaran los prototipos funcionales obtenidos. Por otra parte al ser SARIDIF un subproyecto de investigación de corto alcance que aporta funcionalidades al proyecto COMPUTAPLEX no fue necesario disponer de una exhaustiva especificación de requisitos, más bien se hizo énfasis en respetar los lineamientos de la estructura arquitectónica propuesta tratando de aprovechar los componentes y funcionalidades existentes.

Finalmente, la herramienta Protégé junto con el plug-in Java Bean Generator logra agilizar el proceso de desarrollo puesto que permite la generación de código automático a partir del diseño ontológico planteado, con lo que la integración entre la plataforma COMPUTAPLEX y el código generado resultó una actividad más sencilla de llevar a cabo, siendo entonces el diseño de la ontología la actividad primordial en la que se debe centrar la atención. Un aspecto que produjo problemas en el acto comunicativo entre el agente cliente y el agente publicador estuvo relacionado con la utilización del codec del lenguaje de contenido utilizado, inicialmente se estableció el lenguaje SL pero al obtener problemas en cuanto al uso de dicho codec se decidió hacer uso del lenguaje LEAP, pues se recomienda usar cuando se requiere que los mensajes sean menos densos y más manejables.

7.2. Conclusiones

Actualmente la experimentación en Ingeniería de Software Experimental se encuentra ejecutando experimentos entre laboratorios de varios centros de investigación de manera coordinada, puesto que al contar con la combinación de conjuntos de experimentos permitirá pasar de deducciones aisladas poco generalizables a evidencias globales sobre la construcción de software. Para este fin se han creado infraestructuras de soporte experimental, aunque muchas han quedado en desuso, otras como es el caso de COMPUTAPLEX se encuentran cumpliendo su plan previsto de desarrollo para consolidarse y poder suplir las necesidades existentes de los grupos de investiga-

ción en Ingeniería de Software Experimental.

La plataforma COMPUTAPLEX en cuanto a su estructura tecnológico-arquitectónico se encuentra implementada bajo: la tecnología de agentes inteligentes distribuidos, diseños conceptuales ontológicos, patrones de diseño de software, bases de datos, entre otras. Mediante el desarrollo de los componentes necesarios, se dotó a la plataforma la capacidad para extraer información experimental de hojas de cálculo y ficheros estadísticos SPSS que se encuentran alojados en repositorios distribuidos. Posteriormente, se definió la ontología de acciones del dominio de experimentación en ingeniería de software, la misma que se implementó en COMPUTAPLEX proporcionando la capacidad de interacción capaz de atender las solicitudes emitidas por el agente cliente hacia el agente publicador en cuanto a la extracción de resultados experimentales de los repositorios independientes.

Mediante la utilización del lenguaje XML se pudo evidenciar la amplia variedad de usos que en la actualidad tiene esta tecnología dentro de la construcción de sistemas software, inclusive su importancia dentro de otras tecnologías como son RDF capaces de expresar descripciones de recursos web, permitiendo representar y transferir información de tal manera que se llegue a soportar la interoperabilidad entre diversos repositorios experimentales de sistemas heterogéneos.

El uso de patrones de diseño permiten crear soluciones más genéricas, flexibles y reutilizables. Por medio del mecanismo implementado en el agente publicador mediante el patrón de diseño factory se simplificará en el futuro la agregación de nuevos tipos de fuentes de datos de las cuales se requiera extraer información experimental, permitiendo a COMPUTAPLEX ser más extensible y adaptable a cambios en la medida en que este evoluciona.

Bibliografía

Eduardo Alonso. Ai and agents: state of the art. *AI Magazine*, 23(3):25, 2002.

Fernando Alonso, Loïc Martínez, and Javier Segovia. *Introducción a la Ingeniería del Software: Modelos de desarrollo de programas*. Delta Publicaciones, 2005.

Nicholas M Avouris and Les Gasser. *Distributed Artificial Intelligence: theory and praxis*, volume 5. Springer, 1992.

Fabio Luigi Bellifemine, Giovanni Caire, and Dominic Greenwood. *Developing multi-agent systems with JADE*, volume 7. John Wiley & Sons, 2007.

Martín Solari Buela and SIRA VEGAS HERNÁNDEZ. *PROPUESTA DE PAQUETE DE LABORATORIO PARA EXPERIMENTOS DE INGENIERÍA DE SOFTWARE*. PhD thesis, Universidad Politécnica de Madrid, 2012.

José H Canós, Patricio Letelier, and Mª Carmen Penadés. Metodologías ágiles en el desarrollo de software. *Universidad Politécnica de Valencia, Valencia*, 2003.

Juan Manuel Corchado. 2 modelos y arquitecturas de agente.

Jacques Ferber. *Multi-agent systems: an introduction to distributed artificial intelligence*, volume 1. Addison-Wesley Reading, 1999.

CAI Fernández and Carlos Ángel Iglesias Fernández. *Definición de una metodología para el desarrollo de sistemas multiagente.* PhD thesis, 1998.

Thomas R Gruber. A translation approach to portable ontology specifications. *Knowledge acquisition*, 5(2):199–220, 1993.

Barbara Hayes-Roth. An architecture for adaptive intelligent systems. *Artificial Intelligence*, 72(1):329–365, 1995.

Jim Highsmith. *Adaptive software development: a collaborative approach to managing complex systems*. Addison-Wesley, 2013.

Andreas Jedlitschka and Marcus Ciolkowski. Towards evidence in software engineering. In *Empirical Software Engineering, 2004. ISESE'04. Proceedings. 2004 International Symposium on*, pages 261–270. IEEE, 2004.

Nick R Jennings. Commitments and conventions: The foundation of coordination in multi-agent systems. *The knowledge engineering review*, 8(03):223–250, 1993.

Ernesto Jiménez-Ruiz, Bernardo Cuenca Grau, Ulrike Sattler, Thomas Schneider, and Rafael Berlanga. Safe and economic re-use of ontologies: A logic-based methodology and tool support. In *The Semantic Web: Research and Applications*, pages 185–199. Springer, 2008.

N Juristo and A Moreno. Basics of software engineering experimentation. 2001. Kluwer Academic Publishers.

Youn Hee Kim, Byung Gon Kim, Jaeho Lee, and Hae Chull Lim. The path index for query processing on rdf and rdf schema. In *Advanced Communication Technology, 2005, ICACT 2005. The 7th International Conference on*, volume 2, pages 1237–1240, 2005.

Yannis Labrou. Standardizing agent communication. In *Multi-Agent Systems and Applications*, pages 74–97. Springer, 2001.

Yannis Labrou, Tim Finin, and Yun Peng. Agent communication languages: The current landscape. *IEEE Intelligent systems*, 14(2):45–52, 1999.

Faraón Llorens Largo. *Sistemas de razonamiento y conocimiento distribuido. Agentes inteligentes*. PhD thesis, Universidad de Alicante (UA), 2001.

Ana Mas. *Agentes software y sistemas multiagente: conceptos, arquitecturas y aplicaciones*. Prentice Hall, 2005.

ABM Moniruzzaman and Dr Syed Akhter Hossain. Comparative study on agile software development methodologies. *arXiv preprint arXiv:1307.3356*, 2013.

N Noy and Deborah L McGuinness. Ontology development 101. *Knowledge Systems Laboratory, Stanford University*, 2001.

CA Palacios, CR Soto, and EG Núñez. Cooperación en equipos de agentes expertos. Publicación en extenso de las memorias: XXIV Congreso Internacional de Ingeniería Electrónica, Instituto Tecnológico de Chihuahua, Chih. México.

Shari Lawrence Pfleeger. Albert einstein and empirical software engineering. *Computer*, 32(10):32–38, 1999.

Carlos Reynoso. Métodos heterodoxos en desarrollo de software. *UBA. Argentina*, 2004.

Matías I San Martín and Osvaldo Clúa. Un servicio de movilidad de agentes de software para jade basado en osgi. 2012.

Alejandro G Stankevicius. *Un Modelo Dialéctico para la Deliberación Multiagente*. PhD thesis, Master's thesis, Departamento de Ciencias e Ingenierıa de la Computación, Universidad Nacional del Sur, Bahıa Blanca, Argentina, 2004.

Katia P Sycara. Multiagent systems. *AI magazine*, 19(2):79, 1998.

Sira Vegas, Natalia Juristo, Ana Moreno, Martín Solari, and Patricio Letelier. Analysis of the influence of communication between researchers on experiment replication. In *Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*, pages 28–37. ACM, 2006.

Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in software engineering*. Springer, 2012.

Michael Wooldridge and Nicholas R Jennings. Intelligent agents: Theory and practice. *The knowledge engineering review*, 10(02):115–152, 1995.

Apéndice A

Estructura del archivo de configuración para hojas de cálculo

```
<?xml version="1.0"?>
        <computaplexExcel>
                <excelFile>
                        <fileExcelName>AllData.xls</fileExcelName>
                        <experimentName>
                           <name>UPM-2005</name>
                           <location>0</location>
                        </experimentName>
                        <factors>
                            <factor>
                               <name>Aplicacion</name>
                               <location>4</location>
                            </factor>
                            <factor>
                               <location>9</location>
                            </factor>
                        </factors>
                </excelFile>
                <excelFile>
                        <fileExcelName>AllData2.xls</fileExcelName>
                        <experimentName>
                           <name>UPM-2005</name>
                           <location>0</location>
                        </experimentName>
                        <factors>
                            <factor>
                               <name>App</name>
                               <location>6</location>
                            </factor>
```

```
                                <factor>
                                   <location>7</location>
                                </factor>
                                    <factor>
                                   <location>8</location>
                                </factor>

                            </factors>
                    </excelFile>
            </computaplexExcel>
```

Apéndice B

Código Fuente de Componentes de configuración

ConfigurationExcel.java

```
1
2  package es.upm.grise.computaplex.configuration;
3
4  import java.io.IOException;
5  import java.util.ArrayList;
6  import java.util.HashSet;
7  import java.util.Set;
8
9  import javax.xml.parsers.ParserConfigurationException;
10
11 import org.xml.sax.SAXException;
12
13
14
15 public class ConfigurationExcel {
16
17         private static ConfigurationExcel instance;
18
19         public static ConfigurationExcel getInstance() throws
              ParserConfigurationException, SAXException, IOException{
20
21                 if(instance==null){
22                         return new ConfigurationExcel();
23                 }
24
25                 return instance;
26         }
```

```
27
28          ArrayList<ConfigurationExcelFile> listFilesExcel=new ArrayList
               <>();
29
30
31          public ConfigurationExcel() throws ParserConfigurationException
               , SAXException, IOException{
32                  this("ConfigExcel.xml");
33          }
34
35          //Constructor
36          public ConfigurationExcel(String excelConfigurationFileName)
               throws ParserConfigurationException, SAXException,
               IOException{
37                          ConfigurationExcelFileParser cefp=new
                               ConfigurationExcelFileParser();
38                          listFilesExcel= cefp.parse(
                               excelConfigurationFileName) ;
39          }
40
41          public ArrayList<ConfigurationExcelFile>
               getConfigurationExcelFiles() {
42                  return listFilesExcel;
43          }
44
45  }
```

ConfigurationExcelFile.java

```
1   package es.upm.grise.computaplex.configuration;
2
3   import java.util.ArrayList;
4   import java.util.HashMap;
5
6   public class ConfigurationExcelFile {
7
8           private String excelName;
9           private String experimentName;
10          private int experimentNameLocation;
11          private ArrayList locationList;
12          private HashMap<String, String> factorList;
13
14          public ArrayList getLocationList() {
15                  return locationList;
16          }
```

```
17          public void setLocationList(ArrayList locationList) {
18                  this.locationList = locationList;
19          }
20
21          public HashMap<String, String> getFactorList() {
22                  return factorList;
23          }
24
25          public void setFactorList(HashMap<String, String> factorList) {
26                  this.factorList = factorList;
27          }
28
29          public int getExperimentNameLocation() {
30                  return experimentNameLocation;
31          }
32
33          public void setExperimentNameLocation(int
               experimentNameLocation) {
34                  this.experimentNameLocation = experimentNameLocation;
35          }
36
37          public String getExcelName() {
38                  return excelName;
39          }
40          public void setExcelName(String excelName) {
41                  this.excelName = excelName;
42          }
43          public String getExperimentName() {
44                  return experimentName;
45          }
46          public void setExperimentName(String experimentName) {
47                  this.experimentName = experimentName;
48          }
49  }
```

ConfigurationExcelFileParser.java

```
1
2   package es.upm.grise.computaplex.configuration;
3
4   import java.io.File;
5   import java.io.IOException;
6   import java.util.ArrayList;
7   import java.util.HashMap;
8
```

```
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.w3c.dom.Text;
import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;


public class ConfigurationExcelFileParser  {

        private DocumentBuilder builder;
        private HashMap<String, String> factorList;
    private ArrayList locationsList;

    public ConfigurationExcelFileParser() throws
       ParserConfigurationException{
                DocumentBuilderFactory factory = DocumentBuilderFactory
                   .newInstance();
                builder = factory.newDocumentBuilder();
    }

    public ArrayList<ConfigurationExcelFile> parse(String fileName)
       throws SAXException, IOException{
                File f = new File(fileName);
                Document doc = builder.parse(f);

                Element root = doc.getDocumentElement();
                return getListExcelTable(root);
    }


    private ArrayList<ConfigurationExcelFile> getListExcelTable(Element
        e){
        ArrayList<ConfigurationExcelFile> listaExcelTable = new
           ArrayList<ConfigurationExcelFile>();
                NodeList children =  e.getElementsByTagName("excelFile"
                   );

                for (int i = 0; i < children.getLength(); i++){
```

```
48              Node childNode = children.item(i);
49
50              factorList=new HashMap<>();
51                  locationsList=new ArrayList<>();
52
53                  if (childNode instanceof Element){
54                  Element childElement = (Element)childNode;
55
56                  ConfigurationExcelFile factoresList =
                       getExcelFactorList(childElement);
57                          listaExcelTable.add(factoresList);
58
59                                  }
60          }
61          return listaExcelTable;
62
63      }
64
65
66      private ConfigurationExcelFile getExcelFactorList(Element e){
67          int i=0;
68          ConfigurationExcelFile configExcel=new ConfigurationExcelFile()
               ;
69
70          String fileExcelName=e.getElementsByTagName("fileExcelName").
               item(0).getFirstChild().getNodeValue();
71          configExcel.setExcelName(fileExcelName);
72
73          // Experiment
74          NodeList experiment=e.getElementsByTagName("experimentName");
75          Element experimentElement=(Element)experiment.item(0);
76          if(experimentElement!=null){
77          NodeList experimentChilds=experimentElement.getChildNodes();
78
79          for(i=0;i<experimentChilds.getLength();i++){
80                  Node childNode = experimentChilds.item(i);
81
82                  if (childNode instanceof Element){
83                          Element childElement = (Element)childNode;
84                          String tagName = childElement.getTagName();
85                          Text textNode = (Text)childElement.
                               getFirstChild();
86
87                          String data = textNode.getData();
88
```

```
                        if (tagName.equals("name"))
                                configExcel.setExperimentName(data);
                        if(tagName.equals("location"))
                                configExcel.setExperimentNameLocation(
                                   Integer.parseInt(data));
                }
          }
        }

        //Factores
        NodeList factors=e.getElementsByTagName("factors");
        Element factorsElement=(Element)factors.item(0);
        if(factorsElement!=null){
                NodeList experimentChilds=factorsElement.getChildNodes
                   ();
                for(i=0;i<experimentChilds.getLength();i++){
                        Node childNode=experimentChilds.item(i);
                        if (childNode instanceof Element){
                                Element factor = (Element)childNode;
                                getFactores(factor);
                }

        }

        }

                configExcel.setFactorList(factorList);
                configExcel.setLocationList(locationsList);
                return configExcel;
        }


    private void getFactores(Element e){
                NodeList children = e.getChildNodes();
            String tagName,data;

                for (int j = 0; j < children.getLength(); j++){
                        Node childNode = children.item(j);

                        if (childNode instanceof Element){
                                Element childElement = (Element)
                                   childNode;

                                tagName = childElement.getTagName();
```

```
131                                     data = childElement.getFirstChild().
                                           getNodeValue();
132
133                                     if (tagName.equals("name"))
134                                             factorList.put(String.valueOf(
                                                   locationsList.size()),data);
135                                     else if (tagName.equals("location"))
136                                             locationsList.add(data);
137
138                                 }
139                         }
140
141
142             }
143
144     }
```

Apéndice C

Código Fuente de Componentes de acceso a datos

ExcelData.java

```
1  package es.upm.grise.computaplex.dataAccess;
2
3  import java.io.IOException;
4  import java.io.StringWriter;
5  import java.util.ArrayList;
6
7  import javax.xml.parsers.ParserConfigurationException;
8
9  import org.xml.sax.SAXException;
10
11 import com.hp.hpl.jena.ontology.Individual;
12 import com.hp.hpl.jena.ontology.OntClass;
13 import com.hp.hpl.jena.ontology.OntModel;
14 import com.hp.hpl.jena.ontology.OntModelSpec;
15 import com.hp.hpl.jena.rdf.model.ModelFactory;
16 import com.hp.hpl.jena.rdf.model.Property;
17
18 import es.upm.grise.computaplex.configuration.ConfigurationExcel;
19 import es.upm.grise.computaplex.configuration.ConfigurationExcelFile;
20
21 public class ExcelData implements DataAccess {
22
23  ArrayList<ConfigurationExcelFile> filesConfiguration=new ArrayList<
       ConfigurationExcelFile>();
24
25  public ExcelData() throws ParserConfigurationException, SAXException,
       IOException{
```

```
26    ConfigurationExcel configurationExcel = ConfigurationExcel.
          getInstance();
27    filesConfiguration=configurationExcel.getConfigurationExcelFiles();
28   }
29
30   @Override
31    public String getExperiments(String agentName) {
32     // TODO Auto-generated method stub
33     StringWriter sw = new StringWriter();
34    //FIXME Is this the right place to create the ontology?
35    OntModel m = ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);
36    String expNS = "http://main.grise.upm.es/empiricalStudies#";
37    String coorNS = "http://main.grise.upm.es/coordination#";
38    String agentNS = "agent://" + agentName + "#";
39
40    try {
41     for(int i=0;i<filesConfiguration.size();i++){
42      ExcelDataFile exlsData=new ExcelDataFile(filesConfiguration.get(i))
            ;
43      exlsData.leerArchivoExcel("experiments");
44
45          for(String factores:exlsData.getParametroRetorno()){
46       OntClass o = m.createClass(expNS + "Experiment");
47       Individual experiment = m.createIndividual(agentNS + factores, o);
48         Property name = m.createProperty( expNS + "name" );
49             m.add(experiment, name, factores);
50
51             Property agent = m.createProperty( expNS + "agent" );
52             m.add(experiment, agent, agentNS);
53          }
54  }}
55    catch (Exception e) {
56     OntClass o = m.createClass(coorNS + "Exception");
57     Individual error = m.createIndividual(agentNS + "error", o);
58     Property origin = m.createProperty(coorNS + "origin" );
59     m.add(error, origin, "database");
60     }
61     m.write(sw);
62     return sw.toString();
63          }
64
65  @Override
66  public String getDesignState(String agentName, String nameExperiment) {
67      // TODO Auto-generated method stub
68          return null;
```

```
69  }
70
71  @Override
72  public String getFactorsLevels(String agentName, String nameExperiment)
        {
73  // TODO Auto-generated method stub
74    StringWriter sw = new StringWriter();
75    //FIXME Is this the right place to create the ontology?
76    OntModel m = ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);
77    String expNS = "http://main.grise.upm.es/empiricalStudies#";
78    String coorNS = "http://main.grise.upm.es/coordination#";
79    String agentNS = "agent://" + agentName + "#";
80
81     try {
82          for(int i=0;i<filesConfiguration.size();i++){
83          ExcelDataFile exlsData=new ExcelDataFile(filesConfiguration.get
               (i));
84          exlsData.leerArchivoExcel("levels",nameExperiment);
85
86          for(String factores:exlsData.getParametroRetorno()){
87        OntClass o = m.createClass(expNS + "Experiment");
88           Individual experiment = m.createIndividual(agentNS + factores,
                 o);
89        Property name = m.createProperty( expNS + "valueLevels" );
90           m.add(experiment, name, factores);
91           Property agent = m.createProperty( expNS + "agent" );
92           m.add(experiment, agent, agentNS);
93          }
94          }}
95          catch (Exception e) {
96           OntClass o = m.createClass(coorNS + "Exception");
97           Individual error = m.createIndividual(agentNS + "error", o);
98           Property origin = m.createProperty(coorNS + "origin" );
99           m.add(error, origin, "database");
100         }
101         m.write(sw);
102         return sw.toString();
103         }
104
105 @Override
106 public String getResponseVariables(String agentName, String
       nameExperiment) {
107          // TODO Auto-generated method stub
108          return null;
109 }
```

```
110
111 @Override
112 public String getFactoresPrincipales(String agentName, String
       nameExperiment) {
113   // TODO Auto-generated method stub
114   StringWriter sw = new StringWriter();
115   //FIXME Is this the right place to create the ontology?
116   OntModel m = ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);
117   String expNS = "http://main.grise.upm.es/empiricalStudies#";
118   String coorNS = "http://main.grise.upm.es/coordination#";
119   String agentNS = "agent://" + agentName + "#";
120
121   try {
122    for(int i=0;i<filesConfiguration.size();i++){
123         ExcelDataFile exlsData=new ExcelDataFile(filesConfiguration.get
             (i));
124         exlsData.leerArchivoExcel("factors");
125         for(String factores:exlsData.getParametroRetorno()){
126          OntClass o = m.createClass(expNS + "Experiment");
127          Individual experiment = m.createIndividual(agentNS + factores,
                 o);
128          Property name = m.createProperty( expNS + "namefactors" );
129          m.add(experiment, name, factores);
130
131          Property agent = m.createProperty( expNS + "agent" );
132          m.add(experiment, agent, agentNS);
133   }
134   }
135   }
136  catch (Exception e) {
137   OntClass o = m.createClass(coorNS + "Exception");
138   Individual error = m.createIndividual(agentNS + "error", o);
139   Property origin = m.createProperty(coorNS + "origin" );
140   m.add(error, origin, "database");
141  }
142  m.write(sw);
143  return sw.toString();
144 }
145
146 @Override
147 public String getHypothesis(String agentName, String nameExperiment) {
148 // TODO Auto-generated method stub
149  return null;
150 }
151
```

```
@Override
public String getMetrics(String agentName, String nameExperiment) {
// TODO Auto-generated method stub
 return null;
}

@Override
public String getAnalysis(String agentName, String nameExperiment,
int idHypothesis) {
        // TODO Auto-generated method stub
                return null;
        }

}
```

ExcelDataFile.java

```

package es.upm.grise.computaplex.dataAccess;

import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Set;

import es.upm.grise.computaplex.configuration.ConfigurationExcelFile;
import jxl.Sheet;
import jxl.Workbook;
import jxl.read.biff.BiffException;


public class ExcelDataFile {

    ConfigurationExcelFile descripcionDatos;

        ArrayList<String> parametroRetorno;

        public ArrayList<String> getParametroRetorno() {
                return parametroRetorno;
        }

        public ExcelDataFile() {
                parametroRetorno=new ArrayList<>();
```

```
29          }
30
31          public ExcelDataFile(ConfigurationExcelFile descripcionDatos) {
32                  this.descripcionDatos = descripcionDatos;
33                  parametroRetorno=new ArrayList<>();
34
35          }
36
37          public void leerArchivoExcel(String request) {
38
39                  try {
40
41                          if(request.equalsIgnoreCase("factors")){
42                                  leerFactores();
43                          }
44                          else if(request.equalsIgnoreCase("experiments")
                               ){
45                                  leerExperiments();
46                          }
47
48
49                  } catch (Exception ioe) {
50                          ioe.printStackTrace();
51                          System.out.println("Salta error"+ioe.getMessage
                               ());
52                  }
53
54          }
55
56          public void leerArchivoExcel(String request, String
               experimentName){
57                  if(request.equalsIgnoreCase("levels")){
58                          leerNiveles(experimentName);
59                  }
60          }
61
62
63          public void leerExperiments(){
64                  Workbook archivoExcel;
65                  try {
66                          Set<String> conjunto=new HashSet<>();
67                          archivoExcel = Workbook.getWorkbook(new File(
                               System.getProperty("user.dir")+"/"+
68                          descripcionDatos.getExcelName()));
69                          String data="";
```

```
70                  Sheet hoja = archivoExcel.getSheet(0);
71                          int num=descripcionDatos.
                               getExperimentNameLocation();
72                          for(int i=0;i<hoja.getColumn(num).length;i++){
73                                  data = hoja.getCell(num, i).getContents
                                       ().toString();
74                                  conjunto.add(data);
75                          }
76
77                          for(String datas:conjunto)
78                  parametroRetorno.add(datas);
79
80
81
82                  } catch (BiffException e) {
83                          // TODO Auto-generated catch block
84                          e.printStackTrace();
85                  } catch (IOException e) {
86                          // TODO Auto-generated catch block
87                          e.printStackTrace();
88                  }
89
90
91          }
92
93
94          public void leerNiveles(String experimentName){
95                  Workbook archivoExcel;
96                  try {
97
98
99                          archivoExcel = Workbook.getWorkbook(new File(
                               System.getProperty("user.dir")+"/"+
100                         descripcionDatos.getExcelName()));
101                         String data="";
102                 Sheet hoja = archivoExcel.getSheet(0);
103                         ArrayList ubicacion=descripcionDatos.
                               getLocationList();
104
105                         int locExp=descripcionDatos.
                               getExperimentNameLocation();
106
107                 for (int i=0;i<ubicacion.size();i++){
108
```

```
109                      int num=Integer.parseInt(ubicacion.get(i).toString()
                            );
110
111                    for(int totalFilas=0;totalFilas<hoja.getColumn(num).
                          length;totalFilas++){
112
113                       if(experimentName.equalsIgnoreCase(hoja.getCell(
                             locExp,totalFilas).getContents().toString()))
114                        {
115                               data = hoja.getCell(num, totalFilas).
                                   getContents().toString();
116                                parametroRetorno.add(data);
117                           }
118                    }
119
120                 }
121
122
123                 } catch (BiffException e) {
124                        // TODO Auto-generated catch block
125                        e.printStackTrace();
126                 } catch (IOException e) {
127                        // TODO Auto-generated catch block
128                        e.printStackTrace();
129                 }
130
131
132          }
133
134
135          public void leerFactores(){
136
137                  Workbook archivoExcel;
138                  try {
139
140                         archivoExcel = Workbook.getWorkbook(new File(
                               System.getProperty("user.dir")+"/"+
141                         descripcionDatos.getExcelName()));
142                         String data="";
143                  Sheet hoja = archivoExcel.getSheet(0);
144                         HashMap<String , String> factores=
                               descripcionDatos.getFactorList();
145                  ArrayList ubicacion=descripcionDatos.getLocationList();
146
147                  for (int i=0;i<ubicacion.size();i++){
```

```
148
149                         if(i<factores.size()){
150                                 String clave=String.valueOf(i);
151                                 parametroRetorno.add(factores.get(clave
                                        ));
152
153                         }
154                         else{
155
156                                 int num=Integer.parseInt(ubicacion.get(
                                        i).toString());
157                                 data = hoja.getCell(num, 0).getContents
                                        ().toString();
158                                 parametroRetorno.add(data);
159                         }
160
161                 }
162
163
164                 } catch (BiffException e) {
165                         // TODO Auto-generated catch block
166                         e.printStackTrace();
167                 } catch (IOException e) {
168                         // TODO Auto-generated catch block
169                         e.printStackTrace();
170                 }
171
172                 }
173
174 }
```

Apéndice D

Código Fuente de Componentes de Servicios

AccionesOntology.java

```
1  // file: AccionesOntology.java generated by ontology bean generator.
       DO NOT EDIT, UNLESS YOU ARE REALLY SURE WHAT YOU ARE DOING!
2  package es.upm.grise.computaplex.services;
3
4  import jade.content.onto.*;
5  import jade.content.schema.*;
6  import jade.util.leap.HashMap;
7  import jade.content.lang.Codec;
8  import jade.core.CaseInsensitiveString;
9
10 /** file: AccionesOntology.java
11  * @author ontology bean generator
12  * @version 2014/05/8, 11:56:10
13  */
14 public class AccionesOntology extends jade.content.onto.Ontology  {
15   //NAME
16   public static final String ONTOLOGY_NAME = "Acciones";
17   // The singleton instance of this ontology
18   private static ReflectiveIntrospector introspect = new
        ReflectiveIntrospector();
19   private static Ontology theInstance = new AccionesOntology();
20   public static Ontology getInstance() {
21      return theInstance;
22   }
23
24
25    // VOCABULARY
```

```
26      public static final String OBTENERFACTORES_EXPERIMENTO="experimento
            ";
27      public static final String OBTENERFACTORES_AGENTE="agente";
28      public static final String OBTENERFACTORES="ObtenerFactores";
29      public static final String OBTENEREXPERIMENTOS_AGENTE="agente";
30      public static final String OBTENEREXPERIMENTOS="ObtenerExperimentos
            ";
31      public static final String OBTENERNIVELES_EXPERIMENTO="experimento"
            ;
32      public static final String OBTENERNIVELES_AGENTE="agente";
33      public static final String OBTENERNIVELES="ObtenerNiveles";
34      public static final String NIVEL_EXPERIMENTO="experimento";
35      public static final String NIVEL="Nivel";
36      public static final String FACTOR_EXPERIMENTO="experimento";
37      public static final String FACTOR="Factor";
38      public static final String EXPERIMENTO_NOMBRE="nombre";
39      public static final String EXPERIMENTO="Experimento";
40
41    /**
42     * Constructor
43    */
44    private AccionesOntology(){
45      super(ONTOLOGY_NAME, BasicOntology.getInstance());
46      try {
47
48      // adding Concept(s)
49      ConceptSchema experimentoSchema = new ConceptSchema(EXPERIMENTO);
50      add(experimentoSchema, Experimento.class);
51      ConceptSchema factorSchema = new ConceptSchema(FACTOR);
52      add(factorSchema, Factor.class);
53      ConceptSchema nivelSchema = new ConceptSchema(NIVEL);
54      add(nivelSchema, Nivel.class);
55
56      // adding AgentAction(s)
57      AgentActionSchema obtenerNivelesSchema = new AgentActionSchema(
            OBTENERNIVELES);
58      add(obtenerNivelesSchema, ObtenerNiveles.class);
59      AgentActionSchema obtenerExperimentosSchema = new AgentActionSchema
            (OBTENEREXPERIMENTOS);
60      add(obtenerExperimentosSchema, ObtenerExperimentos.class);
61      AgentActionSchema obtenerFactoresSchema = new AgentActionSchema(
            OBTENERFACTORES);
62      add(obtenerFactoresSchema, ObtenerFactores.class);
63
64      // adding AID(s)
```

```
65
66      // adding Predicate(s)
67
68
69      // adding fields
70      experimentoSchema.add(EXPERIMENTO_NOMBRE, (TermSchema)getSchema(
           BasicOntology.STRING), ObjectSchema.OPTIONAL);
71      factorSchema.add(FACTOR_EXPERIMENTO, experimentoSchema,
           ObjectSchema.OPTIONAL);
72      nivelSchema.add(NIVEL_EXPERIMENTO, experimentoSchema, ObjectSchema.
           OPTIONAL);
73      obtenerNivelesSchema.add(OBTENERNIVELES_AGENTE, (TermSchema)
           getSchema(BasicOntology.STRING), ObjectSchema.OPTIONAL);
74      obtenerNivelesSchema.add(OBTENERNIVELES_EXPERIMENTO,
           experimentoSchema, ObjectSchema.OPTIONAL);
75      obtenerExperimentosSchema.add(OBTENEREXPERIMENTOS_AGENTE, (
           TermSchema)getSchema(BasicOntology.STRING), ObjectSchema.
           OPTIONAL);
76      obtenerFactoresSchema.add(OBTENERFACTORES_AGENTE, (TermSchema)
           getSchema(BasicOntology.STRING), ObjectSchema.OPTIONAL);
77      obtenerFactoresSchema.add(OBTENERFACTORES_EXPERIMENTO,
           experimentoSchema, ObjectSchema.OPTIONAL);
78
79      // adding name mappings
80
81      // adding inheritance
82
83     }catch (java.lang.Exception e) {e.printStackTrace();}
84    }
85    }
```

Experimento.java

```
1  package es.upm.grise.computaplex.services;
2
3
4  import jade.content.*;
5  import jade.util.leap.*;
6  import jade.core.*;
7
8  /**
9  * Protege name: Experimento
10 * @author ontology bean generator
11 * @version 2014/05/8, 11:56:10
12 */
```

```
13 public class Experimento implements Concept {
14
15    /**
16 * Protege name: nombre
17    */
18    private String nombre;
19    public void setNombre(String value) {
20     this.nombre=value;
21    }
22    public String getNombre() {
23      return this.nombre;
24    }
25
26 }
```

Factor.java

```
1
2  package es.upm.grise.computaplex.services;
3
4
5  import jade.content.*;
6  import jade.util.leap.*;
7  import jade.core.*;
8
9  /**
10 * Protege name: Factor
11 * @author ontology bean generator
12 * @version 2014/05/8, 11:56:10
13 */
14 public class Factor implements Concept {
15
16    /**
17 * Protege name: experimento
18    */
19    private Experimento experimento;
20    public void setExperimento(Experimento value) {
21     this.experimento=value;
22    }
23    public Experimento getExperimento() {
24      return this.experimento;
25    }
26
27 }
```

Nivel.java

```
1
2  package es.upm.grise.computaplex.services;
3
4
5  import jade.content.*;
6  import jade.util.leap.*;
7  import jade.core.*;
8
9  /**
10 * Protege name: Nivel
11 * @author ontology bean generator
12 * @version 2014/05/8, 11:56:10
13 */
14 public class Nivel implements Concept {
15
16    /**
17 * Protege name: experimento
18    */
19    private Experimento experimento;
20    public void setExperimento(Experimento value) {
21     this.experimento=value;
22    }
23    public Experimento getExperimento() {
24      return this.experimento;
25    }
26
27 }
```

ObtenerExperimentos.java

```
1  package es.upm.grise.computaplex.services;
2
3
4  import jade.content.*;
5  import jade.util.leap.*;
6  import jade.core.*;
7
8  /**
9  * Protege name: ObtenerExperimentos
10 * @author ontology bean generator
11 * @version 2014/05/8, 11:56:10
12 */
13 public class ObtenerExperimentos implements AgentAction {
```

```
14
15      /**
16    * Protege name: agente
17      */
18      private String agente;
19      public void setAgente(String value) {
20       this.agente=value;
21      }
22      public String getAgente() {
23        return this.agente;
24      }
25
26  }
```

ObtenerFactores.java

```
1
2   package es.upm.grise.computaplex.services;
3
4
5   import jade.content.*;
6   import jade.util.leap.*;
7   import jade.core.*;
8
9   /**
10  * Protege name: ObtenerFactores
11  * @author ontology bean generator
12  * @version 2014/05/8, 11:56:10
13  */
14  public class ObtenerFactores implements AgentAction {
15
16      /**
17  * Protege name: agente
18      */
19      private String agente;
20      public void setAgente(String value) {
21       this.agente=value;
22      }
23      public String getAgente() {
24        return this.agente;
25      }
26
27      /**
28  * Protege name: experimento
29      */
```

```
30      private Experimento experimento;
31      public void setExperimento(Experimento value) {
32       this.experimento=value;
33      }
34      public Experimento getExperimento() {
35        return this.experimento;
36      }
37
38  }
```

ObtenerNiveles.java

```
1
2   package es.upm.grise.computaplex.services;
3
4
5   import jade.content.*;
6   import jade.util.leap.*;
7   import jade.core.*;
8
9   /**
10  * Protege name: ObtenerNiveles
11  * @author ontology bean generator
12  * @version 2014/05/8, 11:56:10
13  */
14  public class ObtenerNiveles implements AgentAction {
15
16     /**
17  * Protege name: agente
18     */
19     private String agente;
20     public void setAgente(String value) {
21      this.agente=value;
22     }
23     public String getAgente() {
24       return this.agente;
25     }
26
27     /**
28  * Protege name: experimento
29     */
30     private Experimento experimento;
31     public void setExperimento(Experimento value) {
32      this.experimento=value;
33     }
```

```
34      public Experimento getExperimento() {
35        return this.experimento;
36      }
37
38  }
```